NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN AUTOMATED APPROACH TO DISTRIBUTED INTERACTIVE SIMULATION (DIS) PROTOCOL ENTITY DEVELOPMENT

by

Michael Canterbury

September 1995

Thesis Advisors: Michael Zyda, John Falby

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
An Automated Approach to Distributed Interactive Simulation (DIS) Protocol Entity Development(U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Canterbury, Michael

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

One problem associated with the *Distributed Interactive Simulation* (DIS) architecture is its limited ability to support real-time, simulated engagements of more than 1000 entities. To solve this problem, it is necessary to refine the existing DIS protocol and optimize the form and content of DIS network traffic. Fundamental to this solution is the need to 1) adopt a structured grammar to be used in describing the protocol, 2) provide a means to author and edit refined DIS data elements, and 3) expedite the coding and implementation of related protocol improvements. In simple terms, the problem addressed by this thesis is to meet each of these requisite needs.

The approach was to design and build a protocol development tool. This was accomplished in three phases. First, a modified Backus-Naur Form (BNF) grammar was formulated for use in modeling DIS data elements. Next, this grammar was applied to the Protocol Data Units (PDU) and data types specified in the current DIS standard. Finally, a tool, the DIS Protocol Support Utility, was developed as a means to automate the process of authoring, editing, and implementing refinements to the DIS protocol.

As a result of this effort, the data elements depicted in the current DIS standard have been specified using a BNF-like grammar. The Protocol Support Utility has been used to process this grammar and automatically generate the program source code associated with each data element, thus expediting the protocol development process.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Distributed Interactive Simulations, networks, communications protocols	128
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

AN AUTOMATED APPROACH TO DISTRIBUTED INTERACTIVE SIMULATION (DIS) PROTOCOL ENTITY DEVELOPMENT

Michael G.Canterbury
Major, United States Marine Corps
B.S., National University, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Author: Michael Gene Canterbury

Michael Gene Canterbury

Approved by: Michael Zyda

Michael Zyda, Thesis Co-Advisor

John S. Falby

John Falby, Thesis Co-Advisor

Ted Lewis

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

One problem associated with the *Distributed Interactive Simulation* (DIS) architecture is its limited ability to support real-time, simulated engagements of more than 1000 entities. To solve this problem, it is necessary to refine the existing DIS protocol and optimize the form and content of DIS network traffic. Fundamental to this solution is the need to 1) adopt a structured grammar to be used in describing the protocol, 2) provide a means to author and edit refined DIS data elements, and 3) expedite the coding and implementation of related protocol improvements. In simple terms, the problem addressed by this thesis is to meet each of these requisite needs.

The approach was to design and build a protocol development tool. This was accomplished in three phases. First, a modified Backus-Naur Form (BNF) grammar was formulated for use in modeling DIS data elements. Next, this grammar was applied to the Protocol Data Units (PDU) and data types specified in the current DIS standard. Finally, a tool, the DIS Protocol Support Utility, was developed as a means to automate the process of authoring, editing, and implementing refinements to the DIS protocol.

As a result of this effort, the data elements depicted in the current DIS standard have been specified using a BNF-like grammar. The Protocol Support Utility has been used to process this grammar and automatically generate the program source code associated with each data element, thus expediting the protocol development process.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

Scientists and engineers have long relied upon modeling as a means to represent the salient characteristics of complex systems or phenomenon. This reliance on surrogate systems (or models) has been of particular importance in cases where the use of the original object under study was not practical for reasons of cost, mass, or complexity. Coincident with the use of modeling, the concept of simulation has evolved. In simple terms, simulation is defined as the process in which the dynamic behavior of one system (the original) can be predicted or extrapolated by observing the behavior of another, less complex system (the model). Simulations provide a means to exercise a given model within a specific context or synthetic environment.

As a predictive tool, the use of modeling and simulation likely dates back to the earliest days of science. It is hard to imagine daVinci exploring human-powered flight without the aid of a model or two. The military, too, has a long-standing history of model and simulation use. From the earliest mock battles to the complex flight simulators of today, the benefits have been immeasurable in terms of cost, training, and advances in tactical planning.

While simulation has been historically a tool of science, digital simulations are a fairly recent phenomenon. First postulated by von Neumann [VNR93], this advance in technology has provided a way to model and simulate larger and more complex entities. The advent of computers has wrought a literal explosion in the number and complexity of computer-based simulators that exist today. Similarly, the range of applications for which simulation is appropriate has continued to grow. Today's military would be ill prepared for the challenges of the tomorrow without the simulators of today.

The size and complexity of a model is generally driven by the size and complexity of the object being modeled. A similar relationship exists in the case of simulation systems. Generally, simulators have been developed to operate within a specific problem domain. A

flight simulator, for example, is engineered to reproduce the environment that a pilot might experience when flying a given aircraft. A weapons simulator would typically emulate the form, fit, and function of an individual weapon platform whether it be a tank, gun, or missile launcher. In each case, the simulator in question is developed to model a single, real-world entity.

Within the realm of stand-alone use, simulation systems have proven invaluable. This is not, however, the context in which wars are fought. A single tank is not dispatched into battle. A single ship does not constitute an armada. Military forces are tasked, organized, and employed as a composite mass of personnel, weapons, and supplies. The diversity, size, and complexity of such a force makes it nearly impossible to accurately model. In the same vein, it is unlikely that a single simulator could be devised to emulate the dynamics of such a force. A more novel approach is required when faced with problems of this ilk.

One solution that has been explored is the concept of distributed or networked simulation. In this approach, the power of many stand-alone simulators can be integrated into a larger, more complex environment. This integration of the various systems is achieved through the use of network technology. In this manner, the complexity of the overall model is spread among the distributed systems. It is this approach that serves as the foundation of the Distributed Interactive Simulation (DIS) effort. [Tarr94]

B. MOTIVATION

The anticipated growth and refinement of DIS dictates the use of an optimized, though flexible, communications architecture. A key component of this architecture is the specification and use of a clearly defined application level protocol. The motivation for this thesis effort is to provide a foundation from which future protocol development might grow. To this end, the scope of this work includes:

- development of a grammar suitable for modeling the current DIS protocol.
- application of this grammar to the DIS data elements specified in the current protocol standard [IEEE93].

- design, development, and implementation of a Protocol Support Utility to be used in protocol development and refinement.

- automated generation of the program source code necessary to implement changes, extensions, and improvements to the DIS protocol.

C. SUMMARY OF CHAPTERS

Chapter II provides an overview of the DIS development effort in terms of its history, purpose, objectives, and direction. Chapter III presents an introduction to the communications architecture associated with DIS, to include an overview of DIS design criteria and related protocols. Chapter IV discusses the current DIS protocol and covers the structure and content of individual Protocol Data Units (PDU). Chapter V presents a more formalized method of defining each PDU using a structured grammar and describes the application of this grammar to the current protocol. Chapter VI discusses the design and implementation of the protocol development tool noted above. Finally, Chapter VII provides a summary of findings, conclusions, and recommendations for follow-on research.

Appendix A contains a sample grammar-based specification of all Protocol Data Units in the existing IEEE standard. Appendix B contains the source listing for the lexical analysis function which is used in parsing any DIS specification given a format similar to that shown in Appendix A. Finally, Appendices C and D provides instructions for the use and maintenance of the utility program.

II. DIS PROGRAM OVERVIEW

The term DIS is used interchangeably to describe an emerging technology, a uniquely defined communications protocol, and an ongoing effort to develop the standards to govern both. While the focus of this thesis relates to DIS as it refers to an applications layer protocol, it is worthwhile to explore the subject in its broader context. To this end, this chapter examines the manner in which DIS has evolved, summarizes the objectives of the DIS effort, and highlights several of the future challenges that must be addressed. Additionally, a summary of other work related to this thesis is presented.

A. EVOLUTION OF DIS

The technology associated with DIS has evolved as a logical extension of the stand-alone simulators of the past. The need to simulate larger and more complex environments has spawned a demand for more capable, robust simulators. Where stand-alone simulators have not been adequate, the concept of networked or distributed simulation has been explored.

1. Simulator Networking (SIMNET)

The first formal requirements for distributed simulation surfaced in the early 1980's. These requirements were based on the need to provide realistic, small unit combat training for U.S Army personnel. While many simulators existed to train individual soldiers and to reinforce their combat skills (drive a tank, shoot a weapon, fly a plane), there were few facilities for the collective training of crews, platoons, or companies. Past experience, both in peacetime and in war, had shown clearly the necessity for training of this type.

In response to the Army requirement, the Defense Advanced Research Project Agency (DARPA) initiated an effort to develop a distributed architecture which would support the use of "virtual" simulations. For training purposes, the term "virtual" simulation inferred a "continuous, real-time, human-in-the-loop" [IST94] environment suitable for the collective

training of certain combat units. The resulting system, developed by Bolt Beranek and Newman (BBN), Perceptronics, and Delta Graphics, was SIMNET.

The principal tenets of the SIMNET system included:

-- a fully distributed architecture with no central event scheduling or control.

-- interconnection of autonomous Simulation Hosts across a Local Area Network (LAN).

-- fully self sufficient host systems, possessing all necessary resources to participate in a given simulation.

-- host responsibility for maintaining information on its own state and communicating the state data and changes to the other hosts on the net.

-- host-to-host communications limited to changes in the host entity state.

-- use of locally executed processes (i.e. Dead Reckoning) to minimize network traffic and communications overhead.

SIMNET was successful in demonstrating the value of the collective training approach. It confirmed the viability of distributed simulation as a platform for such training and provided a foundation for future work in this arena [Mace95].

Today, there are literally hundreds of SIMNET systems in use. It has, in fact, become a defacto (albeit loosely defined) standard in some training communities. SIMNET is not, however, without its deficiencies. First, SIMNET is a single vendor's solution to an Army-specific problem. The entities and models represented in SIMNET reflect a heavy emphasis on land-based, armor-oriented combat. In this regard, SIMNET has minimal utility in the types of training needed by other services (i.e. amphibious landings, ship-to-ship engagements, etc.).

Further, a problem exists in that the SIMNET architecture is based on a specific LAN topology and protocol, Ethernet. This design decision has limited the flexibility of using SIMNET with other LAN topologies [Case90] and has posed added difficulties when attempting to link SIMNET systems across Wide Area Networks (WAN). Admittedly, many of these problems have been addressed and resolved through creative engineering.

Routinely, however, the solutions employed have been at the cost of increased network latency and added processing overhead [Mace95].

Deficiencies notwithstanding, SIMNET remains popular within certain segments of the training community. It has been successfully employed in a variety of crew level training scenarios [Mace90] and has provided users with the capability to establish "virtual" training environments within which multiple participants interact. More importantly, the SIMNET experience has provided a foundation from which future distributed simulation architectures may grow.

2. DIS

Despite the popularity of SIMNET, it was recognized that a more extensible architecture would be needed to meet the anticipated demand for future distributed simulations. Simply put, future distributed simulations would be required to support synthetic environments of greater size and diversity than any previously implemented. An improved infrastructure would be necessary to accommodate the variety of state-of-the-art simulators under development, as well as providing facilities to exploit the broad range of older, or legacy, simulators which had long been in use. In addition to the "virtual" environments hosted under SIMNET, this new infrastructure would be required to support the integration of computer generated forces, and both "live" and "constructive" simulations within the same exercise or environment. By definition, "live" simulations involve the real-time representation of physical weapon systems (i.e. tanks, planes, ships) within the "virtual" environment, while the term "constructive" simulation infers the use of automated "wargaming" systems such as the Marine Air Ground Task Force Tactical Warfare Simulation (MTWS) [IST94].

A major challenge in integrating "live", "constructive", and "virtual" simulations into a single synthetic environment stems from problems in interoperability. As a single-vendor design, interoperability issues were minimized in SIMNET. However, the requirement to interface large numbers of systems which differ in both design and architecture is an

obvious problem from the standpoint of interoperability. Figure 1 illustrates the breath of this problem and depicts the variety of applications and weapon systems platforms targeted for distributed simulation support.

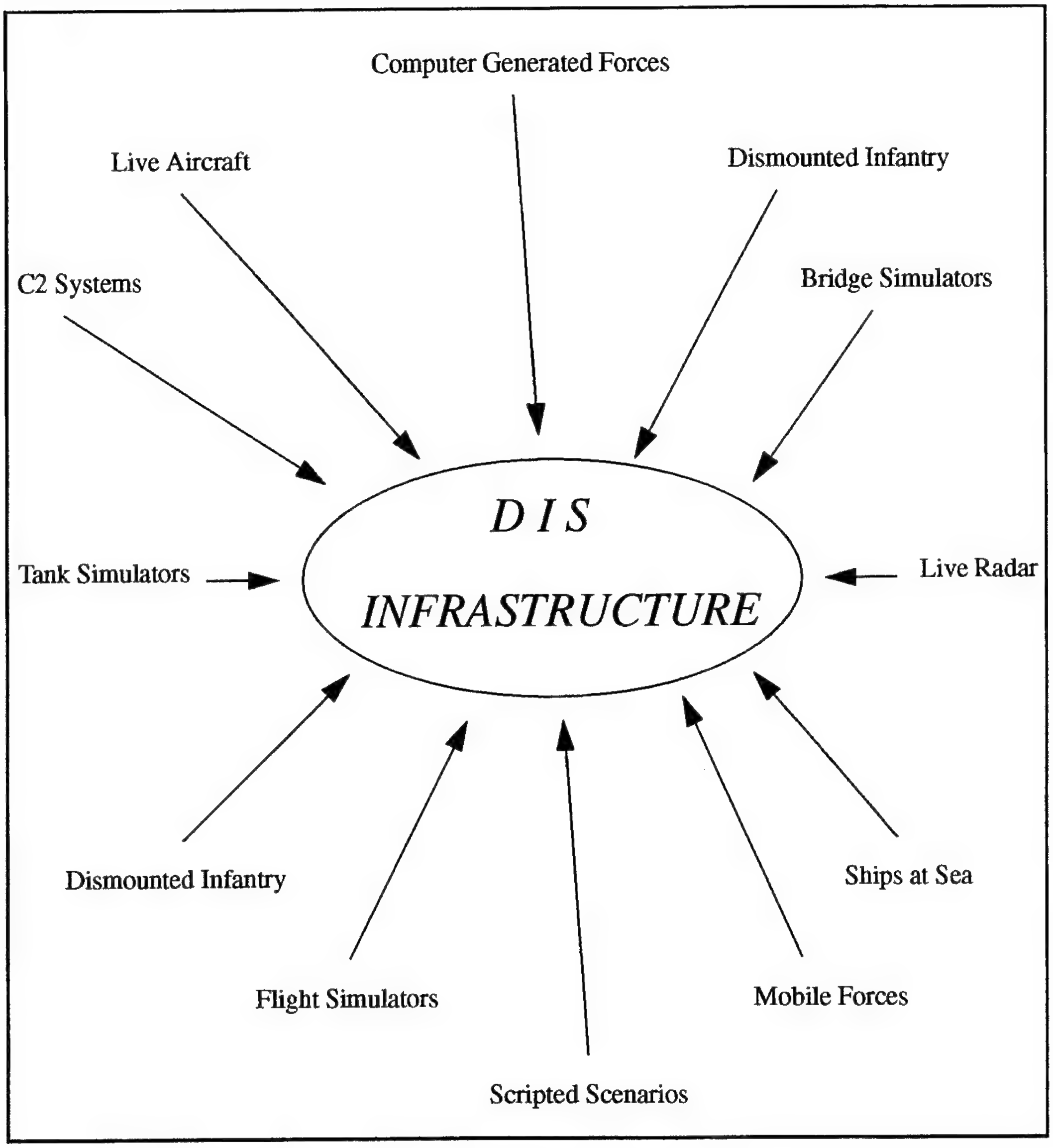

Figure 1. DIS Interoperability Requirements

To address this problem and ensure an adequate level of interoperability among all simulation participants, the need for a "standardized" approach to distributed simulation was recognized. Further, an open architecture was dictated to foster continued advances in

the technology and to stimulate the broader involvement of industry in DIS developments. Finally, a greater degree of flexibility was required to accommodate the growing number and diversity of DIS protocol entities. It was within this mindset that the concept of DIS was borne.

B. DIS PROGRAM OBJECTIVES

The DIS program was initiated in 1989. The intent of this effort is to develop a suitable architecture to support both existing and projected DIS requirements and to produce the policies and standards necessary to facilitate the implementation of DIS systems to meet these needs. Specifically, the principle mission of the DIS effort is to "define an infrastructure for linking simulations of various types at multiple locations to create realistic, complex, virtual 'worlds' for the simulation of highly interactive activities."[IST94] To date, the phrase "highly interactive activities" has inferred the use of DIS facilities for real time, combat-oriented training. The underlying "user requirement" for DIS can best be summarized in a statement by an Army flag officer during preparations for Operation Desert Storm:

> *The toughest thing a commander does at each level of command is to synchronize the battle and make the maximum use of all the different fire support assets available to him. Being good at just one part isn't enough. You have to do it all, and do it in the right sequence with the right timing, or it all starts to get away from you.*
>
> BG James T. Scott
> U.S. Army

And from another source:

> *The services train individual soldiers, sailors, airman, and marines and provide highly trainied combat units and do a very good job. [...But] some things we don't do well. First and foremeost among these is the training and exercise of large, joint, or combined forces to fight on short notice.* [DSBR93]

These sentiments underscore the need for collective training on a much larger scale than that afforded under SIMNET. To this end, the standards and architecture produced under the DIS program are intended as a vehicle to meet this need. With DIS, the modeling, simulation, and training communities have been equipped with the means to produce the type of collective training systems inferred by General Scott and [DSBR93].

Owing to its predominately military emphasis, the current application objectives of the DIS effort include:

-- Joint/Combined Training.

-- Mission Rehearsal.

-- Development/Evaluation of Tactical Doctrine.

-- Battle Reconstruction.

-- Definition of New Combat Systems.

In spite of this particularly military slant, DIS is an excellent example of a “dual use technology”. “Dual use” is used to describe those technologies considered suitable for both military and commercial application. The true potential of DIS is much broader than its military roots would first indicate and its future use may include entertainment, disaster planning and relief, commercial manufacturing, and tele-medicine among others[IST94][Tarr94].

C. IMPLEMENTATION APPROACH

In order to realize the intended objectives noted above, a collaborative effort of government, academia, and industry has been underway to produce the necessary “interface standards, communications architectures, management structures, fidelity indices, technical forums, and other elements necessary to transform heterogeneous simulations into unified seamless synthetic environments” [IST94]. The complexity of this task is apparent. From the management standpoint, it is necessary to achieve some sort of consensus on the many technical issues associated with DIS and encourage both government and industry investment in DIS technologies. From the engineering standpoint,

it is necessary to balance the performance characteristics of the system against the processing and communication resources known to be available. All this must be done within the constraints of the real-time performance demanded by DIS users.

Given this challenge, the DIS community has adopted a "design by informal committee" approach in developing a collection of Industry Consensus Standards to govern DIS. This approach to standards development is somewhat unique within military circles. Even describing the organization is problematic:

> *The structure behind the DIS movement is unique and a bit difficult to describe. There are no articles of incorporation, charters, bylaws, organizational charts, parent organizations, or other elements typical of an organization. What organization there is, is modeled after industry standards development efforts. That is, groups of volunteers gather periodically, do research, debate relevant issues, form consensus, and publish standards. All groups are self-directed and self-governing.* [IST94]

In spite of its somewhat "fuzzy" organizational structure, the DIS effort is not without a central administrative authority. The Institute for Simulation and Training (IST), part of the University of Central Florida, has been tasked with coordinating the DIS standards process. Jointly funded by the U.S. Army's Simulation, Training, and Instrumentation Command (STRICOM) and the Defense Modeling and Simulation Office (DMSO), the IST is responsible for orchestrating the processing and publication of completed DIS standards as well as coordinating the semi-annual workshops which provide a forum for DIS discussions.

Recent interest in DIS can best be gauged by attendance at the semi-annual DIS workshops. The first workshop, convened in August 1989, was attended by more than 50 individuals from government and industry. More recent workshops have attracted more than one thousand participants. As attendance has grown, so has the number of technical working groups staffed by attendees. Each working group is tasked with addressing some specific DIS issue and may spawn "tiger teams" when necessary to resolve key technical problems.Consistent with the overall approach to DIS development, membership in the

working groups is informal and open to any and all interested. At last count, nearly 50 individual working groups and subgroups have been established to handle specific DIS standards issues. The emphasis of current work groups include:

-- Administration	-- Architecture
-- Communication/Security	-- C3I/EW
-- Computer Generated Forces	-- C4I
-- Credible Uses for DIS	-- Distributed Objects
-- Dead Reckoning	-- Data Standards/Repositories
-- Emissions	-- Data Enumeration
-- Fidelity Description	-- Field Instrumentation
-- Interoperability	-- Logistics
-- Protocols	-- Steering Committee
-- Terrain Models	-- Simulation Management
-- Testing	-- Training
-- User Issues	-- Verification and Validation

In spite of its loosely defined structure, the DIS program has been successful in generating a number of standards and guidance documents relative to DIS. Notable among the work produced to date is a collection of documents defining a standard communications architecture for DIS [IST92][IST93a][IST94a], a protocol standard adopted by the Institute of Electrical and Electronics Engineers (IEEE)[IEEE93], and a vision document which presents the problems, goals, and direction of future DIS development[IST94].

D. FUTURE GOALS

In its most recent survey of critical technical issues, the DIS community has enumerated a number of key objectives to guide future research [IST94]. A discussion of each and every objective is beyond the scope of this work. However, several of the research areas cited are pertinent to the subject matter of this thesis in that they relate to the structure, content, and exchange of Protocol Data Units. These include:

1. Functional Area Coverage of PDUs

As the number of DIS participants grows, so will the scope and the type of scenarios that must be supported. It is anticipated that the use of DIS will expand to encompass many new military applications as well as applications in other areas of government and industry.

2. Information Content and Bandwidth Efficiency

In any network application, the conservation of bandwidth is an important issue. DIS is no exception. As the scope of distributed simulations grow and the number of participants increase, so will the demand for additional network bandwidth. This situation dictates the use of a protocol which is generally optimized in terms of bandwidth consumption. To the greatest extent possible, the DIS protocol must incorporate any mechanism available to balance the information content of each PDU with the aggregate bandwidth consumed.

3. Streamline PDUs

One means of achieving a certain degree of bandwidth conservation is to optimize the data elements exchanged across the network. In the case of the DIS protocol, this entails an examination of the form, content, and adaptability of both present and planned PDUs. Changes to the current PDU structure may include elimination of static data fields, compaction of data elements, and the use of tailorable PDUs.

4. Increased Number of Entities

Current ARPA estimates indicate the need to support future simulations with more than 100,000 participating entities [IST94]. It is anticipated that the entity population will include a large contingent of live players and an equally large collection of virtual and constructive simulations...all integrated into a single synthetic environment. The DIS protocol must accommodate the "scalable" nature of future distributed simulations.

5. Optional Communications Profiles

The current profile to support DIS communications is based upon the internet family of protocols (UDP, TCP, IP). This profile provides the communication services necessary to support present day, DIS scenarios. However, as applications grow in size, complexity, and range of use, the suitability of a single profile to render all services in unlikely. A collection of standard profiles will be required, each tailored to a specific application domain [IST94].

6. Enhanced Interoperability

Within the DIS communications architecture, a principle objective is that of achieving and maintaining an exceptionally high degree of interoperability [IST93a]. The DIS approach in this regard is to specify a minimum set of standards to which all DIS-compliant platforms must adhere. The standards include the form and content of PDUs as well as the mechanisms for exchanging data. The ultimate goal is to ensure that existing systems remain compliant while allowing the flexibility necessary for the DIS architecture to mature and evolve.

E. THESIS-RELATED WORK

The specific intent of this thesis is to present a method and mechanism by which current DIS protocol entities may be modeled, manipulated, and potentially improved. It is hoped that this work will provide a foundation from which future protocol improvements might grow. In any event, this effort is offered as a contribution to the ever-growing body of work dedicated to DIS improvements.

While there has been little, if any, past work which directly correlates to the focus of this thesis, there has been a substantial amount of effort aimed at extending the capabilities of DIS. In my view, past DIS work may be classified as "evolutionary", "revolutionary", or included as part of a collection of "tools and utilities".

1. Evolutionary

Owing to the relative infancy of DIS as a technology, the bulk of the documented DIS work has been "evolutionary" in nature. The concept of "evolutionary" work relates to any and all efforts necessary to foster the "maturing" process of a given technology. In this view, the bulk of the DIS development process could be considered an "evolutionary" effort, specifically intended to further DIS technology by means of formulating, publishing, and enforcing standards. Research which would typically be considered "evolutionary" would include proposals to add new DIS entities, adopt new algorithms, and fine tune existing features of the protocol. The Proceedings of the semi-annual DIS workshops are replete with examples of this type of "evolutionary" work. While the volume of "evolutionary" work related to this thesis is sparse, it does exist. For example, in [Prat95], several refinements to the existing protocol were proposed as a means to conserve network bandwidth. The proposed refinements included changes to both the form and content of current DIS protocol entities. The Protocol Support Utility developed as part of this thesis will assist in implementing the type of protocol changes which were proposed.

2. Revolutionary

For the purpose of this thesis, "revolutionary" work is that which significantly alters the course of a given technology. As a relatively new technology, the body of "revolutionary" DIS work is much smaller than that considered "evolutionary". A typical example of this type of work is the adaptation of multicast protocols and the introduction of an Area of Interest Manager [Mace95] concept in DIS. While not directly related to this thesis effort, this work is significant in its approach to bandwidth conservation and DIS scalability. Additional work considered "revolutionary" (though only remotely related to this thesis) would include the introduction of Distributed Object Technology [Peck95], the use of self-describing protocols [Dick94], migratory objects [Felt95], or intelligent agents[Calv94][Wayn95]. In each case, a shift in the overall DIS paradigm would be required, some more radical a shift than others. Given the expected growth of DIS, and the

attendant burden placed upon the communications architecture, the exploration of alternative protocols is a worthwhile endeavor.

3. Tools and Utilities

The final body of work potentially related to this thesis is that of tools and utilities. The Protocol Support Utility developed as part of this thesis is intended as a protocol development aid. To date, the lion-share of tools developed for DIS have focused on real-time data collection and logging, data correlation [Worl95], and simulation management [Milg95]. Additional work has been done in producing tools suitable for simulating DIS network performance [Bate94] and a Scaleability Tools Set has been developed for the construction and generation of DIS scenarios[Vrab93]. It appears that little, if any, past work has been done in formulating tools to support the authoring, editing, and production of DIS PDUs and their associated data elements. As such, this work may provide yet another vehicle for DIS protocol refinement.

III. COMMUNICATIONS ARCHITECTURE for DIS (CADIS)

> ***Distributed Interactive Simulation(DIS).*** *A time and space coherent synthetic representation of world environments designed for linking the interactive, free play activities of people in operational exercises. The synthetic environment is created through real time exchange of data units between distributed,* ***computationally autonomous nodes*** *comprised of entities in the form of simulations, simulators, and instrumented equipment* ***interconnected through standard communicative services****. The computational nodes may be present* ***in one location or*** *may be* ***distributed geographically****.* [IST94a]

Due to its distributed nature, any discussion of DIS must necessarily begin with a look at the communications architecture on which it is based. This chapter presents an overview of the ISO/OSI Reference Model and provides a correlation of DIS functions to the specific layers within this Model. Further, a discussion of the design and performance criteria of the Communications Architecture for DIS (CADIS) is presented.

A. THE ISO/OSI REFERENCE MODEL

The International Organization of Standardization (ISO) has proposed a 7-layer model to be used in describing the computer-communication process. The Open Systems Interconnection (OSI) Reference Model (RM) [ISO84], as it is known, has been broadly accepted as a standard within the data communications community. This model provides a straightforward means to abstract and visualize the different processes and protocols associated with end-to-end, computer communications. Specifically, this tool was "developed as a model for computer-communications architectures and as a framework for developing protocol standards" [Stal94]. The ISO/OSI Reference Model, depicted in Figure 2, is based upon a hierarchy of the following functional layers:

1. Physical Layer

The first, and lowest, layer of the model governs the physical, electrical, and mechanical characteristics of the communications channel or interface. The Physical layer

addresses the procedural issues involved in accessing the physical communications medium and is concerned with the transmission and reception of unstructured bit streams.

2. Data Link Layer

The Data Link layer is principally concerned with the reliable transfer of data across the physical medium. It provides for the blocking (or framing) of data and covers synchronization, error control, and flow control.

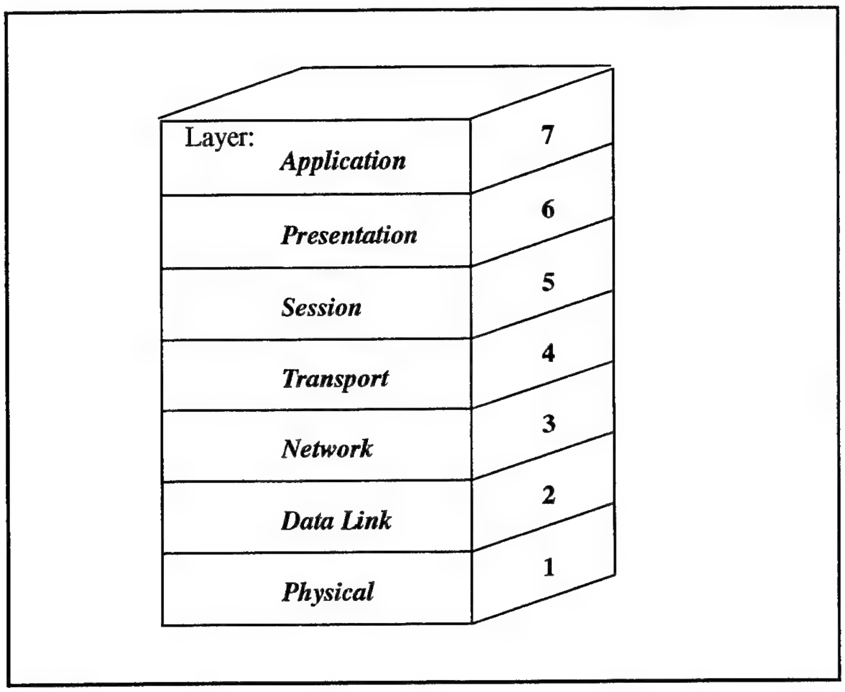

Figure 2. OSI Reference Model

3. Network Layer

The Network layer provides a level of isolation between those layers associated with the physical aspects of the communications process and those concerned with more logical functions. In effect, this layer isolates the upper layers of the model from the lower level switching technologies and electrical interfaces used to connect one system to another. Functionally, the responsibility for establishing, maintaining, and terminating network connections resides at this layer.

4. Transport Layer

While the lower three layers of the model support the physical interface to the transmission path, the Transport layer provides for the management of reliable, transparent, end-to-end connections between connected systems. The responsibility for coordinating error correction and flow control can be found at this layer. By design, all layers residing above the Transport layer operate on the assumption of an error-free communications channel.

5. Session Layer

In order for two applications to exchange information, it is necessary that a logical connection between the applications be established. This connection is termed a "session" and must be managed in a manner similar to the physical connections supported at the lower layers of the model. The Session layer provides the control structure necessary for one application to establish a session and cooperatively communicate with another.

6. Presentation Layer

It is often the case that a session may be established between two applications which, for one reason or another, may represent and store their internal data differently. This difference in syntax is a common occurrence on networks connecting heterogeneous nodes and must be resolved if the reliable exchange of data is to be possible. The Presentation layer provides facilities for syntax transformation and hence, insulates the application processes from potential differences in data representation.

7. Application Layer

The Applications layer, uppermost in the ISO/OSI model, is ultimately responsible for the exchange of information between applications running on connected hosts. This layer provides the interface mechanism (routinely termed an API or Application Programmers Interface) by which a given application can access the communications facilities afforded through the lower layers of the model.

B. CADIS

The communications infrastructure associated with DIS includes functions, services, and protocols found at every layer in the OSI model. Those services which are unique to DIS applications are collectively termed CADIS. By definition, the purpose of CADIS is to "provide an appropriate interconnected environment for effective integration of locally and globally distributed simulation entities" [IST92]. More succinctly, CADIS specifies the communications services and profiles necessary to support a DIS exercise and establishes the required interfaces to the subordinate layer protocols (Transport layer and below) upon which it (CADIS) depends.

1. CADIS Design Philosophy

The CADIS design is driven by the nature of the communication services that must be provided to DIS users. Towards this end, CADIS has three principle areas of emphasis.These include the communication service requirements necessary to support a DIS exercise, the performance criteria needed to meet the real-time demands of DIS applications, and the error detection and synchronization necessary to ensure that the communications subsystem works as intended.

a. Communication Service Requirements

CADIS communication service requirements are tailored to meet the diverse needs of the current DIS community. At present, there are three classes of communication services available to DIS implementors. Within each class are options for either multicast or unicast communication modes.

In the unicast mode, a point-to-point exchange of simulation data is supported. In this approach, a single simulation host may transmit one or more PDUs to a single receiving host. By contrast, multicast services allow the sending host to transmit data to specifically addressed groups of one or more receiving hosts. In simple terms, unicast provides *one-to-one* communication services while multicast provides *one-to-many*.

A broadcast mode of transmission is also provided, though this is in reality a special case of multicast. In the broadcast mode, a single host message is simultaneously transmitted to every host connected to the network. This mode may be best described as a *one-to-all* approach.

From the DIS standpoint, multicast is routinely the transmission mode of choice. In part, the rationale for multicast use lies in the ability to execute multiple DIS exercises simultaneously. This may be done by establishing individual multicast groups for each exercise. Additionally, the use of multicast addressing as a means of reducing DIS bandwidth requirements has been the topic of recent research [Mace90].

The three communication service classes provided by DIS are discussed below.

(1) ***Class 1, Best effort multicast*** - This class of service provides a multicast adaptation of what is known as 'best effort' service. Best effort is, by definition, a connectionless, unreliable communications scheme. By this, it is meant that data packets are forwarded across the network with no assurance or verification that the packets ever arrive at the destination node. No acknowledgments are expected by transmitting nodes, nor are any sent by the nodes receiving data. By eliminating the need to transmit data acknowledgments, network traffic is minimized and system performance enhanced. In a trade-off between performance and reliability, many implementors will opt to conserve network bandwidth at the expense of reliability. This is particularly true in real-time application, and generally the case in DIS.

(2) ***Class 2, Best effort unicast*** - The second communication service class provided under CADIS provides for point-to-point connectivity using the a best effort protocol discussed above.

(3) ***Class 3, Reliable unicast*** - The final communication service class provides for the "reliable" exchange of data. The term "reliable" infers the need for some type of acknowledgment scheme to validate the transmission and receipt of data, and a data

retransmission scheme to resend any data that may be lost or corrupted. This approach is generally employed for synchronization or application level management functions.

b. CADIS Performance Criteria

To ensure the coherence of DIS simulations and the near real-time service demanded by many DIS applications, CADIS defines specific criteria relating to DIS network performance.

(1) ***Latency*** - For performance purposes, DIS applications are classifies as requiring either tightly-coupled or loosely-coupled entity-to-entity interactions. Simulated entities which may be located in close physical proximity to one another would be considered tightly-coupled. In a tightly-coupled scenario, the actions (fire, maneuver, etc.) of one entity must be immediately accounted for by surrounding entities. By contrast, loosely-coupled entities would be those more separated by time or distance and entity-to-entity interactions would be less time critical. A typical tightly-coupled example would be that of fighter aircraft flying in close combat formation. Individual aircraft landing at different airports would require little interaction, hence be loosely-coupled.

Permissible DIS latencies are defined with respect to the ISO/OSI Reference Model discussed earlier in this chapter. **Transport-to-Transport** latency is defined as the time necessary to pass a PDU from the Transport layer service of one Simulation Host, across the network, to the Transport layer service of another. The current CADIS standard specifies an allowable **Transport-toTransport** latency of **300 milliseconds** for ***loosely-coupled*** applications and **100 milliseconds** for the more time critical ***tightly-coupled*** scenarios [IST95].

A second CADIS-specified latency standard is that of **Transport-to-Physical** latency, the time necessary for simulation data to pass from the Transport layer services to the Physical layer services of a given Simulation Host. An allowable **Transport-to-Physical** latency of **10 milliseconds** is cited under CADIS for both *tightly-* and *loosely-coupled* applications [IST95].

*(2) **Bandwidth*** - DIS bandwidth requirements are exercise dependent and not specifically established under CADIS. The required exercise bandwidth may be affected by the number entities simulated and the mixture and type of entities in a given scenario. Also, security overhead and the choice of dead reckoning algorithms may have an impact on the demand for DIS bandwidth [IST93a].

*(3) **Error Detection and Synchronization*** - Error detection in DIS refers to the ability to detect corrupted PDUs. CADIS relies upon the checksum mechanisms of lower layer protocols (TCP and UDP, discussed below) for this facility. No error correction features are established under CADIS.

The means of synchronizing DIS exercises is application dependent. CADIS allows systems developers the latitude to implement the synchronization mechanism which best fits the needs of the application domain. Past implementations in this area have include the use of the Global Positioning System (GPS) and other external time sources.

2. Related Protocols

As noted earlier, the upper layer protocols assume that an error free communications channel exists from DIS node to DIS node. By virtue of this assumption, the DIS infrastructure is heavily reliant on lower layer protocols to establish and maintain the required connectivity. Additionally, DIS relies upon these protocols as sole means of error detection, as noted above. Today, most DIS applications support and employ the Internet Protocol Suite for Transport and Data Link layer services. This family of protocols is comprised of:

a. Transmission Control Protocol (TCP)

TCP is the Transport layer protocol used for the "reliable" exchange of simulation data under DIS. As mentioned earlier, "reliable" transmission schemes require the use of some sort of acknowledgment mechanism to ensure that data is successfully transmitted and received. Also, a retransmission mechanism is required to rebroadcast any

data that is lost or corrupted in the communications process. Obviously, the benefit of "reliability" is at the expense of protocol complexity and processing overhead.

As a "reliable", connection-oriented, point-to-point protocol (i.e. unicast), TCP is well suited for applications that require reliability over performance. Within DIS, TCP use is generally limited to simulation management functions [IST95]. The TCP protocol is formally specified in a standard issued by ISO (RFC 793) and the design and use of TCP and other internet protocols is covered in substantial detail in [Rose91].

b. User Datagram Protocol (UDP)

UDP is another Transport layer protocol and is an alternative to TCP. In contrast to TCP, UDP is not considered "reliable". It has no facility for acknowledgments nor data retransmission. If messages are lost or corrupted during the communications process, no action is taken. By removing the overhead associated with reliability, a more streamlined protocol, UDP, is possible. UDP is a connectionless, best effort approach which can support multicast transmission modes and is formally specified in a standard issued by ISO (RFC 768). For most DIS applications, reducing network traffic to conserve bandwidth and minimize latencies is more important that the type of "reliability" afforded by TCP. Hence, UDP is generally the protocol of choice.

c. Internet Protocol (IP)

IP is the Network layer protocol on which CADIS relies. During message transmission, the purpose of the IP protocol is to encapsulate the Transport layer data in an envelope (IP datagram) that is suitable for transmission across the network. During the receive process, the envelope is "opened" and the Transport layer data is passed to the appropriate Transport layer protocol.

As would be expected, IP supports both TCP and UDP protocols and provides the source and destination addressing needed to pass simulation data from one Host to another. The IP protocol is formally specified in an ISO-issued standard (RFC 791).

3. DIS Protocol Suites

Initially, the communications architecture of DIS was to evolve in three phases. Each phase was to be distinguished by the collection of communication services and protocols on which it was based. In this manner, DIS would evolve from the protocol suites commonly found in use today, to the protocols which had been targeted (or mandated) for the future.

The first phase was to be based on the internet protocol family discussed earlier. This protocol was considered mature by industry standards and would provide a suitable starting point for DIS implementations. In fact, it is this protocol suite that is predominately used in DIS today.

In the second phase of this evolution, the DIS architecture would migrate to a family of OSI protocols which were considered compliant with the interface standards published by ISO. Theoretically, the OSI protocol suite was considered a more robust approach to distributed communications than that offered under the internet family. However, the OSI suite had not been completely standardized and was not considered to be a mature collection of protocols.

The final phase was to migrate DIS to a fully GOSIP-compliant class of protocols. The GOSIP or Government Open Systems Interconnection Profile protocol family is a mandated U.S. government adaptation of the OSI suite. Similar to its OSI counterpart, the GOSIP protocols were not yet mature nor suitable for full scale DIS use.

While the concept of successively migrating from one protocol suite to the next was sound in theory, it was somewhat lacking in application. The OSI and GOSIP protocol suites were not widely embraced in industry and, as such, were evolving at a slow pace. The internet suite had achieved broad-based acceptance and many implementors were comfortable with its use. More importantly, the DIS community recognized that the range of potential DIS applications seemed to grow without bound. As the range of applications grew, so did the diversity in demands placed upon the communications infrastructure. Given this scenario, the adoption of a single protocol suite for DIS use might

unintentionally limit the size of the DIS application domain. Further, it is unlikely that a single family of protocols would be capable of providing communication services for every potential implementation of DIS.

As an alternative to the phased implementation of specific protocol suites, the concept of multiple communication *profiles* was proposed [Lope94]. Under this proposal, DIS implementations would not be limited to a single protocol suite. Instead, a collection of standard profiles would be approved for DIS use. This approach offers a great deal more flexibility in meeting future DIS demands. The number of profiles is expected to grow as the use of DIS becomes more widespread. This proposal was only recently adopted within the DIS community and to date, only two profiles have been approved. Within each profile, three classes of services (as discussed earlier in this chapter) are provided. The current profiles are intended for "traditional LAN/WAN" environments and include:

a. Profile-1: Internet-Broadcast

This profile is based upon the internet protocol suite and establishes three classifications of service:

-- Class 1: Best Effort multicast with broadcast addressing.

-- Class 2: Best Effort unicast using UDP/IP.

-- Class 3: Reliable unicast using TCP/IP.

b. Profile-2: Internet-Multicast

This profile is similarly based on the internet family and support the following classes:

-- Class 1: Best Effort multicast using UDP over IP/MC.

-- Class 2: Best Effort unicast using UDP/IP.

-- Class 3: Reliable unicast using TCP/IP.

C. DIS-TO-ISO/OSI CORRELATION

The principle value of the OSI Reference Model is as a means of abstraction. A given communications architecture may be distilled into a series of clearly defined system functions, and each function may be mapped to an appropriate layer in the model. By this means, even the most complex architectures may be reduced to a more understandable form. The DIS architecture may be similarly modeled.

1. Standards

Figure 2 illustrates the relationship between specific layers of the Reference Model and the variety of standards that apply to DIS communications. The documents applicable to this discussion include site dependent LAN standards, Transport/Network layer protocol standards, and those standards which apply uniquely to DIS.

a. DIS Standards

The DIS Standards are, in fact, a family of standards and draft documents which serve as guidelines for DIS implementation and use. These standards govern the protocol [IEEE93][IST93b], the communications architecture [IST92][IST94a], and the data types and values [IST93] associated with DIS. As depicted in Figure 3, the DIS standards generally apply only to the upper three layers of the ISO Model. This makes sense in that the upper layers (Session, Presentation, and Application) are concerned specifically with those functions associated with the host-to-host exchange of applications data. By definition, DIS is considered to be an applications layer protocol. This definition specifically refers to the definition and exchange of DIS PDU...clearly an application layer function. Not surprisingly, the DIS protocol is defined in a single, application level standard [IEEE93] which establishes the form and content of individual protocol elements. Similarly, the DIS enumeration data (a Presentation Layer issue) is set forth in a separate, but complimentary standard [IST93] which defines the acceptable range of values that each data element in DIS may assume.

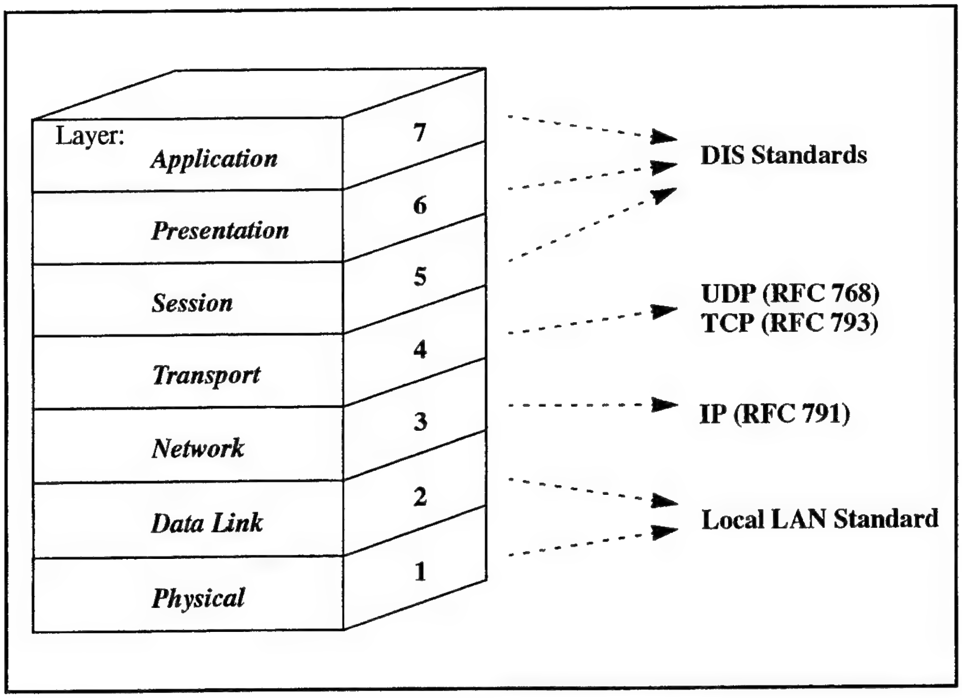

Figure 3. DIS-to-OSI Standards Correlation

b. RFC 768/RFC 793/RFC 791

The Request For Comment (RFC) series of standards are published by ISO for international consideration. The three RFC standards associated with DIS govern the Transport and Network layer protocols (discussed previously) on which DIS relies.

c. Local LAN Standard

The Local LAN Standards govern the low level interface to the transmission media. The most widely used of these standards are issued by the Institute for Electrical and Electronic Engineers (IEEE) and cover a variety of LAN-oriented protocols (Ethernet, Token Ring, etc.).

2. Functional Correlation

Layer Number	Layer Name	DIS Content
7	Application	Type of data exchanged (positional, orientation, etc.). Rules for determining effects of events (e.g. collision). Remote Entity Approximation (Dead Reckoning).
6	Presentation	PDU format and interpretation. Representation of position, orientation, units, and encoding.
5	Session	Procedure for starting, stopping, joining and exiting an exercise.
4	Transport	Source and destination process-to-process addressing. Packet Assembly/disassembly, if required. Ordering, if required. Reliability, if required.
3	Network	Source and destination host addressing.
2	Data Link	Framing onto physical link. Logical link control. Medium access and conflict resolution.
1	Physical	Physical characteristics of the medium.

Table 1. DIS-ISO/OSI Functional Correlation

DIS is considered to be an applications protocol. In this context, the term application refers to services provided in the upper three layers of the ISO/OSI model. This sort of simplification of the OSI model is common. Table 1 depicts the correlation between specific DIS functions and the corresponding layers of the Reference Model [IST95][Zes93].

Frequently, the protocols associated with the Session, Presentation, and Application layers of the model will be collectively referred to as a "process" or "applications" layer protocol [Stev90]. For the purpose of this thesis, the term "applications layer protocol" will be used when discussing the form, content, and use of DIS PDUs.

IV. DIS APPLICATIONS PROTOCOL

By design, the CADIS standard [IST94a] defines a communications subsystem across which real-time simulation data may be carried. As was noted in the last chapter, the scope of CADIS is limited to the specific communication services and profiles which are supported under DIS. A second, and equally important part of the DIS communications infrastructure is the Applications Protocol. This chapter provides an overview of this protocol and of its principle data elements.

A. PHILOSOPHY

The *Standard for Information Technology, Protocols for Distributed Interactive Simulation* [IEEE93] establishes the form and content of the individual data entities that must be exchanged during a given DIS exercise. The purpose of this standard is to provide for the common identification and representation of DIS data items. Further, the standard addresses the manner in which these data entities may be collected into a meaningful message, the PDU. Finally, this standard dictates the protocol to be used by exercise participants in exchanging DIS PDUs and defines the key algorithms (e.g. dead reckoning) [IST94] that must be implemented in each Simulation Host.

The DIS protocol standard (DIS 1.0) was first adopted by the IEEE on 17 March 1993 [IEEE93]. Subsequent updates to this directive have been produced as draft standards, and are generally referred to as the DIS 2.x protocol series. As of the date this thesis was prepared, a more current version of the IEEE standard is scheduled to be released based upon the DIS 2.1 draft.

The revised standard, IEEE 1278.1, reflects recent refinements made in the protocol and associated data entities. Specific improvements include added facilities to support:

-- simulated radio communications, both voice and tactical data link.

-- characterization of electromagnetic emissions as part of electronic warfare scenarios.

-- added Simulation Management features.

The revision of the DIS protocol standard is indicative of the manner in which DIS is expected to evolve. As new requirements surface, the protocol is to be extended to meet the new demands. As deficiencies are encountered, the protocol will be refined and corrected.

B. DESIGN OBJECTIVES

To large extent, the design objectives of the DIS protocol have been inherited from its predecessor, SIMNET. These objectives reflect the decisions and compromises made to meet the competing demands inherent in a DIS distributed environment...real time performance, minimal latencies in entity interaction, and efficient bandwidth utilization. The design principles on which the DIS protocol is based include [IST94]:

1. Autonomy of Simulation Hosts

Every DIS participant is required to broadcast pertinent information about itself to all other participants in the exercise. Receiving nodes will determine the applicability of any messages that they receive and will assess the impact of any event on the entity that they represent. Sending nodes have no responsibility in determining the impact of their actions, only reporting that the action has occurred. In this way, no single entity is dependent upon another to exist in the simulated environment.

2. Object/Event-based Architecture

In this approach, a simulated entity is required to possess and maintain information on every other entity involved in the exercise. The information about static or non-changing objects (e.g. fixed terrain features) is generally provided as part of the exercise initialization data. Dynamic entities are expected to notify all other participants of their movements or activities as they occur. This notification process involves the transmission and reception of PDUs.

3. Minimized Transmission of State Change Information

One approach in minimizing the demand for network bandwidth is to restrict the frequency at which entities provide updates on their status within the exercise. Under DIS,

participating entities need only transmit update to other entities when a change in their last reported state has occurred. Given no change in state, updates are limited to a predetermined time interval (generally 5 seconds) and merely serve as an "I'm still alive"-type message.

4. Use of Dead Reckoning Algorithms

A second avenue in minimizing the demand for bandwidth is the use of dead reckoning algorithms. In this approach, each entity maintains a dead reckoning "model" of itself and every other dynamic entity in the exercise. This model is used to extrapolate the changing position of a moving entity between required state updates. Basically, the model is used to project the path and rate of movement of the modeled entity over a given period of time. The modeled entity will constantly monitor the accuracy of its own dead reckoning model and compare the model's predictions with the entities "actual" position. If the projection of the model exceeds a predefined error threshold, a PDU is issued across the network. In the absence of this update, the other participants in the exercise will use the model's output to establish the position and movement of the modeled object.

The use of dead reckoning algorithms minimizes the need for state updates due to changes in an object's position. As long as the movement of a given entity is accurately predicted, only periodic update PDUs will be generated and received.

5. Obligation to Transmit "Ground Truth"

It is expected that every participant in the exercise is an honest player. The coherency of the simulated environment is tied to accuracy of the data exchanged between participants.

C. PROTOCOL DATA UNITS (PDUs)

Within a DIS exercise, Protocol Data Units serve as the "lingua franca" or common language by which participating simulators may communicate. Collectively, DIS PDUs define the form and content of information that is to be passed between DIS hosts. The use

of standard data messages, ala PDUs, tends to isolate the communications process from the design or architecture of individual connected nodes. At the implementation level, it matters little what processor a given host may contain, or what LAN is used to link simulation participants. The important aspect in this process is that every host use the same message format and rules for exchanging simulation data. This is the essence of the DIS protocol and rationale behind the concept of a PDU.

In DIS, state changes and any event information pertaining to an exercise is communicated through the use of a rigidly specified PDU. The current protocol standard defines twenty-seven different PDUs, each tailored to satisfy a particular DIS communications requirement.

The PDUs defined in the current standard are organized into five application-related families. The following depicts each family and its associated PDUs:

1. **Entity Information/Interaction Protocol Family**

-- Entity State PDU -- Collision PDU

2. **Warfare Protocol Family**

-- Fire PDU -- Detonation PDU

3. **Logistics Protocol Family**

-- Service Request PDU -- Resupply Offer PDU

-- Resupply Receive PDU -- Resupply Cancel PDU

-- Repair Complete PDU -- Repair Response PDU

4. **Simulation Management Protocol Family**

-- Create Entity PDU -- Remove Entity PDU

-- Start-Resume PDU -- Stop-Freeze PDU

-- Acknowledgment PDU -- Action Request PDU

-- Action Response PDU -- Data Query PDU

-- Set Data PDU
-- Data PDU
-- Event Report PDU
-- Message PDU

5. Distributed Emission Regeneration Family

-- EM Emission PDU
-- Designator PDU
-- Transmitter PDU
-- Signal PDU
-- Receiver PDU

Of the PDUs defined above, only four are specifically intended to describe entity interaction. The Entity State PDU, as its name would suggest, is used to communicate state information about a given entity. This information generally includes position, movement, and appearance. The Fire and Detonation PDUs are used to describe the characteristics of simulated weapons/ordnance and their effect when used. Finally, the Collision PDU is used to convey physical contact between simulated entities.

The remaining PDUs are used primarily for simulation management or when other aspects of a combat environment (logistic support or tactical communications) are to be simulated.

D. PDU STRUCTURE

From the interoperability standpoint, the principle role of the DIS protocol is to establish "a standard for entity definition and entity communication" [IST94]. To this end, the protocol standard [IEEE93] dictates the structure and use of each DIS PDU. As an example, Table 2 depicts the form and content of the Entity State PDU (ESPDU). The table reflects the size and composition of the DIS data elements which constitute the ESPDU. These data elements (as well as all others used in building PDUs) are likewise defined in the protocol standard.

ESPDU Data Element	Data Fields	Field Size (bytes)
PDU Header	Protocol Version Exercise ID PDU Type Protocol Family Time Stamp PDU Length Padding	(1) (1) (1) (1) (4) (2) (variable)
Entity ID	Site Application Entity	(2) (2) (2)
Force ID		(1)
# Articulation Parameters	n	(1)
Entity Type	Entity Kind Domain Country Category SubCategory Specific Extra	(1) (1) (2) (1) (1) (1) (1)
Alternate Entity Type	(same as Entity Type)	(8)
Linear Velocity	X,Y,Z Components	(12)
Location	X,Y,Z Components	(24)
Orientation	Psi, Theta, Phi	(12)
Appearance		(4)
Dead Reckoning (DR)	DR Parameter Record	(40)
Entity Markings	Entity Marking Record	(12)
Capabilities		(4)
Articulation Parameters (AP)	AP Record	(n * 16)

Table 2. Structure and Content of the Entity State PDU

The form, size, and content of the Entity State PDU is typical of that found in the other twenty-six PDUs defined in the current protocol standard.

As illustrated in Table 2, PDUs may be of variable length (note the Articulation Parameters). Whether of fixed or variable length, PDUs tend to be quite large. In general, PDU size and transmission rates have a direct relationship to the bandwidth needed to support a DIS exercise. Recognizing this fact, the CADIS standard recommends that PDUs not exceed 1400 octets (bytes) in length [IST95].

The issue of PDU size and information content becomes particularly important when one considers simulated scenarios of the future in which more than 100,000 entities must interact [IST94]. Recent estimates indicate that to support an exercise of this magnitude, each participant using the DIS existing protocol would require bandwidth of nearly of 375 million bits-per-second (Mbps) [Mace95]. This is nearly four times the capacity provided by the fastest LANs (e.g. FDDI Token Ring, Fast Ethernet, etc.) of today [Stal94]. It is not likely that near-term advances in communications science will afford this level of performance. In the absence of some revolutionary leap in communications technology, other means of "scaling" the DIS protocol must be found if it is to be used in future distributed simulations.

V. DIS DESCRIPTIVE GRAMMAR

At the application layer, the data structures exchanged by protocol entities are potentially much more complex. Therefore, it is necessary to introduce a new formalism for describing these structures.

This new formalism is termed an "abstract syntax,"...

Marshall T. Rose [Rose91]

The stated goal of this work is to introduce both a method and a mechanism to accommodate future efforts in DIS protocol development and improvement. In part, the intent is to provide a means by which new or improved DIS protocol entities (PDUs and data elements) may be formally described or modeled. Subsequently, the protocol model is to be automatically translated into the programming language source code necessary to implement the redefined protocol. This chapter discusses the first steps in this process, the formulation and application of a descriptive grammar for DIS use.

A. TAXONOMY

The approach taken in preparing this thesis was a three phased endeavor. The first phase was to formulate a descriptive grammar, or abstract syntax, which could be used to model DIS PDUs and related data elements. The second phase of effort involved applying the formulated grammar to the protocol entities in the current DIS standard to produce a structured specification of the protocol. The final phase, discussed in the next chapter, involved the development and implementation of an automated tool to process the grammar-based specification of DIS.

B. STRUCTURED GRAMMAR

The benefit of a structured grammar (or abstract syntax) lies in its ability to simplify the representation of a complex problem or concept. Much like the ISO/OSI Reference Model, a grammar may be used as a method of abstraction. Formal grammars, of one type or another, are commonplace in every field of science. Most often, these grammars are used

as a means to better understand or manage things that would otherwise be unmanageable or impossible to fathom.

Within the field of computer science, grammars are fundamental to compiler design, automata theory, and the study and implementation of programming languages [VNR93]. Likewise, structured grammars are routinely used as a tool in data communications to describe the formal rules (or protocols) needed to support the exchange of information between communicating nodes. Given the widespread use of grammars in similar applications, the use of a structured syntax to describe the DIS protocol seems reasonable.

The use of a formal grammar was not the only abstraction method considered in describing DIS. In fact, several means of modeling the protocol, both graphical and grammar-based, were explored. The graphical techniques (e.g. Finite State Machine models) were rejected because of the difficulties anticipated when ultimately translating the graphically modeled protocol into the source code needed for host system implementation. By contrast, parsing and translating a grammar-based syntax is a fairly straightforward process. In spite of the adage that "*a picture is worth a thousand words*", a grammar-based approach was chosen for this effort in protocol modeling.

Due likely to the manner in which DIS has evolved, no formal specification of the protocol appears to exist. As such, the selection of a particular grammar to model DIS was considered an important issue. Preliminary investigation revealed several grammars considered suitable for DIS use, most notably, Abstract Syntax Notation 1 (ASN.1).

ASN.1 is an abstraction grammar developed specifically as an application layer syntax. It is formally specified in an ISO standard [ITU88] and is used as a tool to define the structure, content, and management of application layer entities (PDUs and data elements) [Rose91]. At first glance, ASN.1 appeared to be the perfect choice as a descriptive grammar for DIS. Some tout ASN.1 as "*the network programming language of the 90's*" [Rose91] and equate its future popularity with that of the "C" programming language in the 80's. However, the same sources admit that the "*bells and whistles of ASN.1*

lead to unnecessary complexity" and that "*prudence*" generally "*dictates a minimalist approach be taken*" when modeling communications protocols.

For the purpose of this thesis, the grammar chosen to model DIS was required to possess the following attributes:

-- sufficiently flexible to model all DIS data elements.

-- minimally complex in terms of readability and use.

-- easily parsed.

-- generally accepted for use in similar applications.

ASN.1 was suited for this type of application. It possessed the necessary constructs to describe every DIS PDU and data element. However, initial efforts to model DIS using ASN.1 clearly demonstrated its tendency towards "*unnecessary complexity*". Its syntax proved cumbersome and the degree of complexity was such that it would probably discourage use of the grammar by others in future DIS development efforts. While ASN.1 would be suited for a rigorous, more formal definition of the protocol, it was not considered appropriate for the simple needs of this thesis effort.

As an alternative to ASN.1, a Backus-Naur Form (BNF) [VNR93] grammar was considered. BNF is a metalanguage typically used as a formal method for defining program language constructs. BNF is widely accepted as a descriptive syntax and has been in use since the earliest days of ALGOL in the late 50's and early 60's. More importantly, it may be easily adapted to meet most any application need.

In terms of complexity, BNF is the embodiment of simplicity. The grammar is based upon the fundamental concept of *syntactic units* and *terminal* symbols. Syntactic units are the grammar constructs which are considered valid in the particular language being described. In BNF, syntactic units are generally enclosed in angular brackets...

< *grammar construct* >

This notation is used for clarity and as a means to distinguish the acceptable grammar constructs from other BNF symbology.

The terminal symbols used in BNF are, in effect, the primitive or atomic language elements on which the described grammar is based. Similar to the manner in which letters of the alphabet are used to create words and sentences, terminals in BNF are used to build the syntactic units of a given grammar.

The symbol ‘ ::= ‘ is used in BNF to signify metalinguistic equivalence. It is used to separate the left and right sides of a production rule or definition.

Ultimately, the grammar chosen for use in this thesis was a modified version of BNF, influenced somewhat by ASN.1 syntax. The syntactic constructs used to build the DIS descriptive grammar are shown in Table 3, below. Note the addition of extra brackets, ‘ << >> ’, to distinguish PDUs, braces, ‘ {} ‘, to signify dynamic data structures (generally implemented using pointers*), and square brackets, ‘ [] ’, to indicate a sized array of elements. Also, a semi-colon, ‘ ; ‘ is used to indicate the end of each DIS entity definition.

Grammar Construct	Meaning
<< pdu >>	PDU
< composite >	composite data element
{< composite >}	composite structure*
< composite >[size]	composite element array
atomic	atomic data element
{ atomic }	atomic structure*
atomic[size]	atomic element array
enum8, uint16,...	alias for primitive data type
::=	“is defined as”
;	“end definition”

Table 3. DIS Descriptive Grammar Constructs

C. APPLICATION

Figure 4 illustrates the use of the DIS descriptive grammar in modeling a typical PDU, a composite data element, and an atomic data type. Appendix A provides a structured specification of all data elements in [IEEE93].

```
<<EntityState_PDU>>  ::=    <PDUHeader>
                            <EntityID>
                             ForceID
                             num_ap
                            <EntityType>
                            <Alt_EntityType>
                            <Entity_Linear_Velocity>
                            <Location_Entity>
                            <Entity_Orientation>
                             entity_appearance
                            <DR_Parameters>
                            <Entity_Marking>
                             capabilities
                            <Articulat_Params>[num_ap]
                              ;

<PDUHeader>          ::=      protocol_version
                              exercise_id
                              PDUType
                              protocol_family
                              time_stamp
                              length
                              padding16
                              ;

ForceID              ::=      enum8;
```

Figure 4. DIS Descriptive Grammar Example

The primitive data types (e.g. enum8) expressed in the grammar are, in reality, aliases for the basic data types found in a typical programming language (C/C++). In the Protocol

Support Utility developed as part of this thesis, the correlations depicted in the following Table apply.

Grammar Primitive	C/C++ Data Type
bool32	unsigned int
enum8	unsigned char
enum16	unsigned short
float32	float
float64	double
pad8	char
pad16	short
pad32	unsigned int
uint8	unsigned char
uint16	unsigned short
uint32	unsigned int

Table 4. DIS Grammar-to-Data Type Correlation

As an application note, it is worthwhile to mention the difficulties involved in modeling the DIS protocol using this (or any other) descriptive grammar. The principle problem lies in the lack of clearly specified naming conventions for individual DIS entities.

By their nature, application layer protocols are ambiguous [Rose91]. The lack of naming conventions compounds this situation and makes automated code generation problematic, if not impossible. Routinely, a compiler would treat the following as unique data elements:

Entity_State_PDU
EntityState_PDU
EntityStatePDU

The problem becomes more acute if the above declarations are to be used as data types in a given implementation. The "C" language functions:

int doThis(PDUType pdu); and **int doThis(PDU_Type pdu);**

are clearly different and would be treated as such by the compiler. The adoption of naming standards associated with the DIS protocol could do much to reduce this potential for ambiguity.

In spite of the naming issue, the application of the descriptive grammar to the protocol was a straightforward process. The syntax was unambiguous in identifying which constructs were PDUs, which were Composite data types, and which were Atomics. Capitalization was used as a means of discriminating which Atomic types were meant to be formal data type definitions (ala **typedef** in "C"). In this approach, the Atomic declaration, **PDUType**, would signify a formal data type definition, while the declaration: **num_params**, would indicate an Atomic data element defined in terms of some primitive data type associated with the programming language used for DIS implementation.

Given a structured specification of the protocol, it is possible to develop tools to manipulate the definition of the protocol by altering its grammar-based description. The modified description may then then used to regenerate the program code needed to implement the altered protocol in a given host system. This is the underlying premise on which the design and implementation of the Protocol Support Utility is based.

VI. PROTOCOL SUPPORT UTILITY

Formal Description Techniques (FDTs) and automated tools have long promised to make substantial contributions to the development of communications protocols. Have they?....Yes; but, the success is more often based on the capability to automate a tool than it is on the use of FDTs,....

Harry Rudin[Rudi92]

One advantage of a structured grammar is the ease in which it may be parsed and processed. This is particularly true of the descriptive grammar just presented. This chapter discusses the development and implementation of the Protocol Support Utility, a tool to complement the grammar proposed in the last chapter and serve as a resource in future efforts to refine the DIS protocol.

A. RATIONALE

As noted earlier in this work, the concept of DIS "scalability" is a growing concern. The size and complexity of future simulations dictate that every possible step be taken to optimize the current protocol, minimize system latencies, and conserve available bandwidth. To this end, the DIS community has identified a number of protocol-related improvements as objectives for future research (briefly discussed in Chapter II). Of the many objectives cited, several serve as a foundational focus for this thesis effort. These include:

-- Steps to increase functional areas covered by PDUs.

-- Balancing PDU information content with bandwidth efficiency.

-- Streamlining PDUs.

-- Definition of a tailorable PDU set.

-- Development of additional communications profiles.

Each of the noted objectives can be achieved, at least to some degree, by manipulating the form and content of individual PDUs. Removal of static, redundant, or infrequently used data from specific PDUs aids in reducing demand for bandwidth. New functional

areas, as well as added communications profiles, can be more rapidly developed and implemented with a set of easily modified (or tailorable) PDUs. Experimentation with new or improved PDUs might be encouraged if an automated means of protocol development were available.

The Protocol Support Utility is intended as both a proof of concept and as a protocol development resource. As a proof of concept, the Utility demonstrates the viability of automated support tools for use in DIS protocol development. As a development resource, the Utility provides a means to easily manipulate the form and content of a given PDU, and automatically generate the program source code necessary to implement any changes made to DIS protocol entities. This Utility, and the methods associated with its use, are specifically intended to contribute to the growing body of work dedicated to the improvement of the DIS Applications Protocol.

B. FUNCTIONAL OVERVIEW

Simply put, the Protocol Support Utility provides an automated facility for protocol entity development, to include authoring, editing, and implementation. As depicted in Figure 5, the input to the Utility is a grammar-based description of one or more protocol entities. Using the edit functions of the Utility, the form or content of any protocol entity may be altered. This is done by merely changing the grammar that describes the particular entity in question. Additionally, new entities may be built by using the descriptive grammar constructs presented earlier in this work. Finally, the Utility may be used to automatically generate the source code needed to implement any changes made in the protocol as a result of this process.

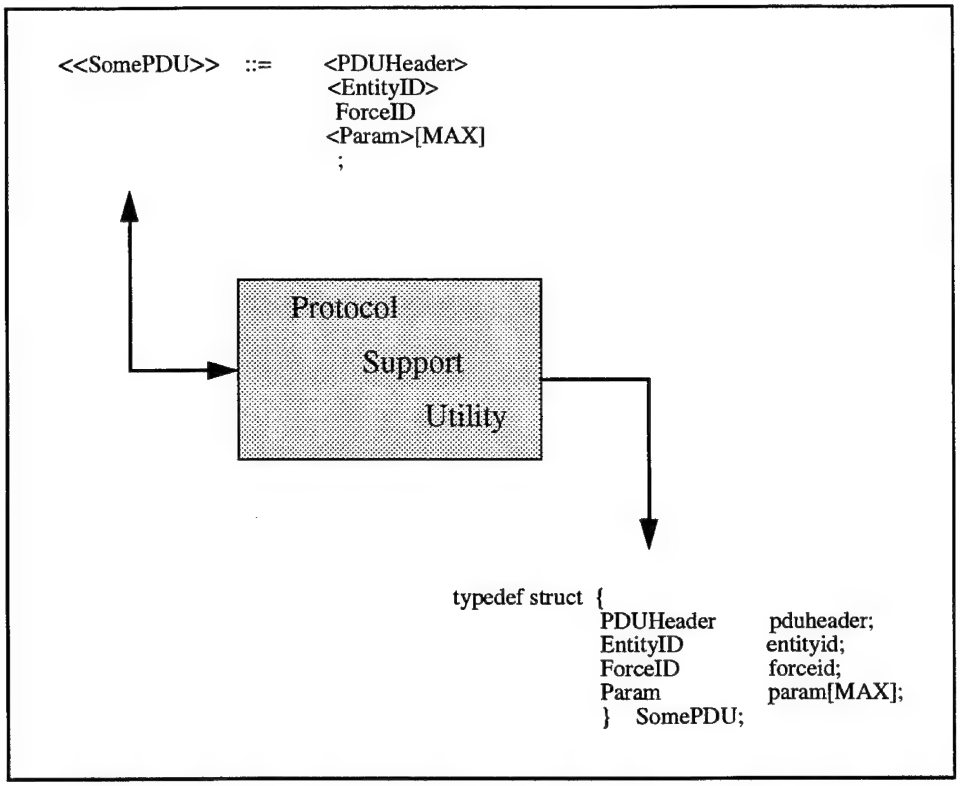

Figure 5. Protocol Support Utility Functional Overview

C. ASSUMPTIONS AND CONSTRAINTS

In the development of a typical computer-based application, it is prudent to establish any assumptions or constraints which may impact the design and use of the system. In building this tool, it was necessary to make several assumptions relating to the architecture of the Utility and the operating environment in which it is intended for use. Additionally, a number of assumptions were required with regard to the DIS protocol and the entities on which it is based.

1. Operating Environment

As a DIS protocol development tool, the Utility is intended for use on any DIS-capable system. Specific requirements are a UNIX-compliant host woth OSF/Motif. The platform of choice for this implementation was Silicon Graphics.

2. Performance Criteria

The Protocol Support Utility is a stand-alone, non-real-time development tool. It is assumed that there are no time constraints placed upon system performance.

3. Basic Protocol Characteristics

From the protocol standpoint, it is assumed that every PDU will contain a PDU Header as a means of identification. The format of the PDU Header is assumed to be consistent with that established in the current DIS standard [IEEE93]. The grammar representation of the PDU Header is depicted in Appendix A.

Additionally, it is assumed that the "Other PDU" entity will be defined by default in any protocol which is developed using this Utility. Routinely, this PDU is used as a means of identifying any PDUs which are considered "unknown" or not otherwise defined as part of the DIS standard.

Finally, protocol entities are assumed to be either PDUs, Composite data elements, or Atomic data elements. Composite data elements are, in effect, composite data structures. The Composite elements are those protocol entities (other than PDUs) which are comprised of one or more different data elements. By contrast, an Atomic data element is defined in terms of a singular, primitive data type as discussed in Chapter V of this work.

4. Entity Naming

The Protocol Support Utility must parse and process a grammar-based representation of the protocol. As such, the Utility is heavily reliant on the naming conventions applied to DIS protocol entities. In the absence of a published naming standard, the following is assumed:

--- PDU naming will consist of the name of the PDU followed by the letters "PDU" (i.e. EntityStatePDU).

--- Any protocol entity which is to be implemented as a data type definition (i.e. typedef) shall have a name containing one or more capital letters (i.e. PDUType). This primarily relates to the definition of Atomic data elements and is necessary to facilitate

proper source code generation. Those Atomic elements which are not intended as defined data types shall be represented by lower case names or labels (i.e. num_params).

5. Arbitrary Numeric Constraints

The internal data structures within the Utility have been implemented as fixed size arrays (discussed below). As such, it was necessary to assume certain arbitrary limits on the number of individual protocol entities which the Utility can support during a given session. Programmatically, these values are defined in the global header file (**disPSU_GLOBAL_DEFINES.h**) of the Utility. As presently implemented, the following numeric constraints have been assumed:

---	Total symbols supported	1000
---	Maximum PDUs	50
---	Maximum Composite/Atomic Types	200
---	Maximum Elements in a PDU/Composite	75

6. Applicability

It was also assumed that the principal role of the Protocol Support Utility is that of a DIS data definition tool. Based upon a given grammar, the tool is expected to generate the source code definitions for every protocol entity so described. This tool is not intended as an automated means for reengineering the entire DIS communications infrastructure. Such a task is beyond the scope of this Utility. This does not, however, preclude broader use of this tool given a suitable DIS network interface and host system implementation.

D. DESIGN AND IMPLEMENTATION

As previously depicted in Figure 5, the Protocol Support Utility provides an automated facility to translate a structured representation of the DIS protocol into the C/C++ language source code necessary to implement the PDUs and data elements described. To achieve this goal, the Utility must provide the following capabilities:

--- Read and parse a given DIS descriptive grammar.

--- Maintain and store the grammar in an internal representation suitable other program use.

--- Support both authoring and editing of the protocol grammar as necessary.

--- Support source code generation based upon specific Application Profiles.

--- Support interactive use of the tool through a simple to use, graphical user interface (GUI).

The general design of the Protocol Support Utility mirrors this list of required capabilities. The Utility was implemented using both the "C" and "C++" programming languages, and the following describes the major functional areas of the application.

1. Tables and Structures

The heart of the Protocol Support Utility lies in the data structures established to maintain and store the protocol grammar. Implemented as a series of fixed sized arrays, the internal Tables serve as a repository for the individual protocol entities which are defined in a given grammar. Four separate tables have been implemented to accommodate the DIS grammar constructs.

a. Symbol Table

The ***Symbol Table*** is the single structure in which the character strings, or labels, associated with each grammar symbol are stored. In addition to this label, each entry in the ***Symbol Table*** contains three index fields. Each index field corresponds to one of the protocol constructs (PDU, Composite, or Atomic) supported by the grammar.

Only a single index field within a given ***Symbol Table*** entry is used. If the label which has been defined corresponds to a PDU, only the PDU index field is updated and the other two index fields (Composite and Atomic) remain UNDEFINED. If the label corresponds to an Atomic data element, only the Atomic index field is used. In effect, the index fields in each ***Symbol Table*** entry provide a way to *remember* the type of protocol entity associated with each label. The value of a given index is determined by the order in which protocol entities are stored in one of the other Tables discussed below.

b. PDU Table

By default, the ***PDU Table*** is a 50 element composite array. Each entry in the ***PDU Table*** represents a single PDU definition and consists of a label index (pointing at its name in the ***Symbol Table***), an element count, and an array of one or more protocol elements (either Composite or Atomic). The specific number of elements contained in a given PDU is indicated by an integer value stored in the element count field.

c. Composite Table

The ***Composite Table*** is based upon a structure identical to that used for the ***PDU Table***. Similar to a PDU, a Composite element may be comprised of one or more protocol elements (either Composite or Atomic). The label (character string) associated with a Composite element is stored in the ***Symbol Table*** and is referenced in a manner similar to that used for the ***PDU Table***.

d. Atomic Table

The ***Atomic Table*** differs slightly from the tables used to store PDUs and Composite entities. An entry in the ***Atomic Table*** consists of two ***Symbol Table*** indices and a single character string. The first index into the ***Symbol Table*** references the position at which the name of the Atomic element is stored. The second index provides the name of the primitive data type associated with the Atomic element. The character string in each entry is used during the generation of source code. Its purpose is to hold a string representing a programming language data type (i.e. unsigned int).

The remaining data structure used to support the DIS grammar is the ***element***. In the discussion above, it was noted that a PDU or Composite construct may consist of one or more protocol elements. The ***element*** data structure represents a single instance of a protocol element (either Composite or Atomic) as it occurs within a PDU or Composite entity definition. The complete "C" language declaration for the ***element*** structure, and each data Table, may be found in the **genCLASS_globals.h** file provided as part of the Utility.

Collectively, these data structures are used to store the entire grammar-based description of a DIS protocol. However, the problem of parsing a grammar file input and properly loading each data table has yet to be addressed. This information is presented in the section that follows.

2. Lexical Analysis

Based upon the syntax discussed in the last chapter, the Protocol Support Utility uses a LEX-based scanner for parsing an input grammar. This scanner is designed to recognize the unique delimiters which identify PDUs and Composite or Atomic data elements.

Basically, the scanner has been designed as a simple Finite State Machine. As may seen in Appendix B, the scanner states include:

< C_CMMT >

< CPP_CMMT >

< INITIAL >

< PDU_DEF >

< STRUCT_DEF >

The first two states depicted above are provided to support embedded comments within a grammar specification. Both "C" and "C++" commenting conventions are allowed. In reality, the scanner detects a comment and ignores all subsequent tokens until it recognizes an appropriate comment-ending delimiter.

The next state, < INITIAL >, is the static or reset state. In this state, the scanner is awaiting the definition of a protocol entity. This state corresponds to the left-hand side of a production rule and the scanner will accept any DIS grammar construct.

If, while in the < INITIAL > state, the scanner detects a PDU or Composite construct, the < PDU_DEF > or < STRUCT_DEF > state will be set as appropriate. In either of these states, the scanner expects to encounter only Composite or Atomic constructs. The < PDU_DEF > and < STRUCT_DEF > states correspond to the right-hand side of a typical production rule.

When the scanner detects an allowable construct, it invokes one of the Table Management functions associated with the Utility. These functions provide the means to initialize, load, and update the internal data tables (Symbol, PDU, etc.). The most notable among these functions include:

Function	Purpose
void initTABLES();	Initialization of all internal Tables.
void stripToken();	Remove grammar delimiting symbols.
void addSymbol();	Add token to Symbol Table.
void addSymbolPDU();	Add PDU element to PDU Table
void addSymbolCOMP();	Add Composite element.
void addSymbolATOM();	Add Atomic element.

The appropriate functions are invoked as each grammar construct is read and recognized. As such, the internal data tables will be fully loaded when the scanner detects the end of the input grammar file. The lexical analysis process is used repeatedly during program operation as a method of initializing and updating the grammar stored in the internal data tables.

3. User Interface

The Protocol Support Utility is operated by means of a windows-based, graphical interface, shown in Figure 6. Implementation of this interface was completed using OSF/ Motif [Ferg93] text widgets. The interface consists of a Main Program Window which supports a pulldown menu bar, a grammar view window, and a display for runtime status messages. Also supported are a series of buttons which launch the individual editors for each different type of protocol entity (PDU, Composite Type, Atomic Type).

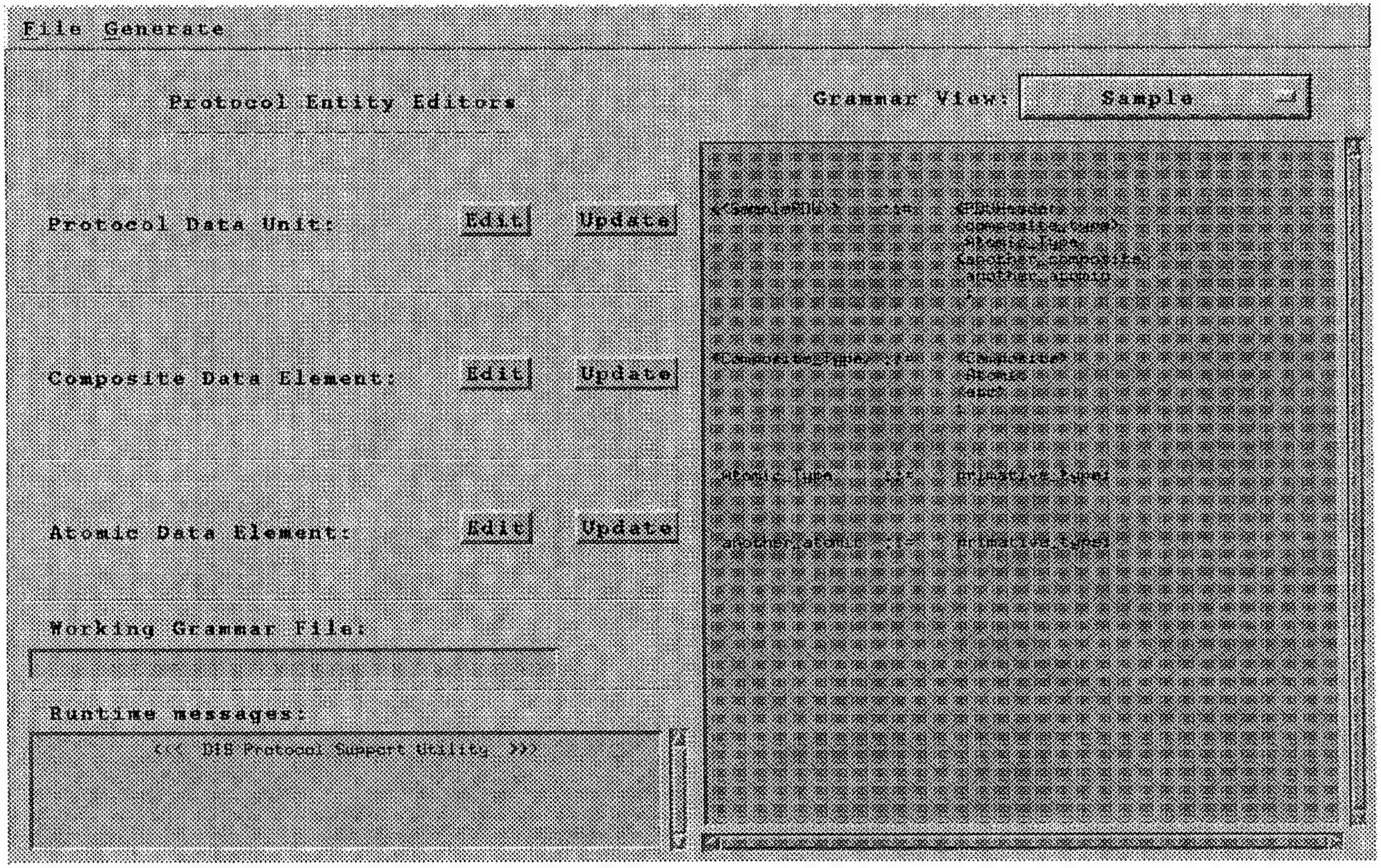

Figure 6.Protocol Support Utility User Interface

a. Grammar View Window

The **Grammar View** window provides a facility to view the grammar-based representation of DIS protocol elements. This is implemented as a display-only window and editing of the display is not supported.

The contents of the viewing window may be selected by means of the pulldown widget labeled "**Grammar View**". This widget is positioned immediately above the **Grammar View** window and offers the following selections:

-- **Sample** - Allows viewing of a sample grammar specification. Selection of this option invokes the **viewSampleCB()**.

-- **PDU** - Allows viewing of all PDUs resident in the PDU Table. Selection of this option invokes the **viewPDUCB()**.

-- **Composite** - Allows viewing of all Composite elements resident in the Composite Table. Selection of this option invokes the **viewCompositeCB()**.

-- **Atomic** - Allows viewing of all Atomic elements resident in the Atomic Table. Selection of this option invokes the **viewAtomicCB()**.

-- **All** - Selects viewing of all PDUs, Composite and Atomic types defined. Selection of this option invokes the **viewAllCB()**.

b. Runtime Message Window

The ***Runtime Message*** window is positioned in the lower portion of the Main Program Window. This window displays the program's runtime status and error messages as they occur. This window was implemented using a Motif Scrolling Text widget and, like the ***Grammar View*** window, is used for display only.

c. Working Grammar File Window

This window displays the filename of the grammar file currently being processed. This window is implemented using a simple, single line Text widget. The ***Working Grammar File*** window is for display purposes only and may not be edited.

A single function is used to display information in all test windows. The

void msgPrint();

function is used to initiate the display of a given string within a particular window. The parameters passed to this function include the string to be displayed and the target window (Widget) desired. As characters are displayed in a given widget, this function is also used to update and maintain the respective cursor location within the window. This is necessary for character insertion purposes under Motif.

4. Grammar Editors

Silicon Graphics provides an ASCII-based text editor, **jot**, for default use on most SGI workstations. The Protocol Support Utility uses this program as the principle means of editing the protocol grammar.

The Utility has been designed to support separate edit sessions for PDUs, and Composite and Atomic data elements. The editor is "launched" by depressing the "**Edit**" button associated with a particular protocol entity. Once an edit session is initiated, a **jot** window will appear. The **jot** display will contain the protocol elements which correspond to the particular protocol entity to be edited. The specific contents of the **jot** window will reflect the protocol elements currently stored in the associated internal data table.

Actual editing of the grammar may be done as one would edit any text file under **jot**. In fact, any text-based editor may be used to edit a grammar file, though only **jot** has been integrated into this Utility. The criteria for selecting the **jot** editor was based upon its availability, adequacy for use, and ease of implementation. An alternate editor may be added if required, and the procedure to do so is described in Appendix D.

To some extent, the other windows within the display may be used during an edit session. If desired, information in the ***Grammar View*** window may be "Copied and Pasted" into the **jot** window. This is particularly useful in building PDU's which are based upon previously Composite or Atomic entities.

Once editing is completed, the revised grammar may be saved and the **jot** session terminated. The result of this process is a grammar scratch file which contains the edited data. This file is used in the subsequent **Update** process to be discussed later.

Because of the reliance on **jot**, implementation of the **Edit** feature required little more than a process to extract the needed information from the internal tables, create a scratch file for the edit session, and invoke **jot** by means of a UNIX **system** call. The program callbacks employed in the **Edit** process include:

void pduEditCB ();

void compEditCB ();

void atomEditCB ();

The **Update** function must be invoked to apply any grammar changes to the internally stored data. This is accomplished by depressing the **Update** button associated with a preceding edit session.

The purpose of the **Update** feature is to apply changes to the internal tables which correspond to the edited grammar. Normally, this would be a complicated process and would require that the information stored in the internal data tables be compared with the contents of the previously saved scratch file. This approach was considered to be needlessly complex and a potential source of significant processing overhead. Instead, an alternate approach using the lexical scanner is employed.

As part of the **Update** process, an temporary input grammar file is built using the scratch file created during a prior **Edit** process. Appended to this input file is the internal table data for the "non-edited" protocol entities. The resultant file contains the altered grammar from the edit session and the original, unchanged table entries.

The final step in the **Update** chain is to reinitialize the internal tables and invoke the scanner to process the temporary file built in the step above. As the scanner processes the file, the internal tables are reloaded with the updated grammar. The specific callback functions implemented as part of the Update process include:

void pduUpdateCB ();

```
void compUpdateCB ();
void atomUpdateCB ();
```

5. Source Code Generation

The ability to author and edit the description of individual protocol entities is of marginal value if there is no way to translate the protocol grammar into a more usable, readily implemented form. One of the principle features of the Protocol Support Utility is its ability to generate program source code based upon a previously parsed grammar. Because of the way in which protocol entities are modeled using the descriptive grammar, the Utility is particularly well suited for producing the source code data definitions (i.e header files).

Given the diversity of the DIS community, it is unlikely that the source code produced for one DIS application would meet the needs of another. Even with the adoption of a global naming convention for protocol entities, substantial differences in host system implementations would exist. One system may be minimally impacted by changes to the protocol while another may be substantially dependent on the form and content of each protocol entity.

A single method of source code production is not appropriate for DIS purposes. Hence, the need for the Protocol Support Utility to be flexible in terms of the structure, content, and scope of the source code produced. As a means to this end, the concept of an Application Profile has been incorporated into the tool.

An Application Profile is a collection of functions needed to produce the desired source code output for a given system. These Profiles may differ widely. One may result in generation of an entire object-oriented network harness produced in C++. Another Profile may generate a single Ada procedure. The characteristics of a Profile are dictated by the implementation dependencies within the host system application and the extent to which automated source code generation is possible.

Two Application Profiles (NPSNET, Class-based) have been incorporated into the Protocol Support Utility. Appendix D discusses the straightforward manner in which additional Profiles may be added.

a. NPSNET Profile

NPSNET is a DIS-compliant, networked software architecture built to support large scale virtual environments [Mace94]. This system was selected as a candidate profile based upon its local availability and its maturity as a host platform for DIS use.

The first step in developing an Applications Profile is the identification of DIS-related implementation dependencies within the system in question. In building the NPSNET profile, this was done by identifying the specific functions within the network harness which were dependent upon the definition of protocol entities, whether PDUs or protocol data types.

A cursory inspection of NPSNET revealed that nearly thirty percent of the network interface software was in some way dependent upon the protocol definition. The majority of dependent functions were those that merely referenced a data type defined by the protocol (i.e. PDUType). This dependency was easily resolved by ensuring that the naming conventions used in both the DIS grammar and the NPSNET application were consistent.

Six functions within the NPSNET network harness were wholly dependent on the protocol. Due to specific implementation dependencies, only two of the protocol-dependent functions were candidates for automated source code generation. The remaining functions were purely host-dependent implementations and not suited for code generation using this Utility.

Once the implementation dependencies had been isolated, the Profile was developed. NPSNET network functions which had little or no dependence on the protocol

definition were placed in source code template files (shown below) to be used in regenerating the NPSNET network software.

__disbridge.cc **(NPSNET DIS_bridge)**

__disnetlib.cc **(NPSNET DIS_net_manager)**

The implementation dependent functions, those not suited for code generation, were placed in separate template files to be used "as is" during source code production. These files are shown below.

__disbridge_depend.cc

__disnetlib_depend.cc

Finally, the functions needed to generate the remaining source code for the NPSNET Profile were implemented. In this particular Profile, the actual source code generated by the Utility was limited to two functions within the network harness and a global header file used to define all protocol entities. The specific functions responsible for source code generation (under the NPSNET profile) can be found in **_npsPROFILE.C** file supplied as part of this Utility.

b. Class-based Profile

The second profile included with the Utility is the Class-based Profile. This is meant to demonstrate the ability of the Utility to generate source code based upon different programming constructs or paradigms. The same NPSNET network harness was used as a basis for this profile.

Under this Profile, each PDU is implemented as a derived class based upon the **dis_bridge** and **dis_net_lib** classes implemented under NPSNET. A virtual **write_pdu()** function is included in each definition.

The program code generated under this Profile is purely notional and intended as a demonstration in the production of object-oriented source code. In its current state, it is not intended as a means to generate an entire Class-based network harness.

E. INSPECTION AND TESTING

Testing of the Protocol Support Utility was performed by means of exercising the program and evaluating the subsequent results. The lexical analysis function was validated by scanning a typical grammar file and programmatically examining the contents of the internal data tables.

The user interface was evaluated in terms of both function and performance. All display functions were verified by ensuring that each window contained the appropriate user data.

Finally, source code generation was verified by means of visual inspection and host system recompilation. In the case of the NPSNET Profile, the source code generated by the Utility was compared to the original system source code on a line-by-line basis. Following the visual inspection, recompilation of the network harness was performed using the Utility-produced source code. To achieve a successful recompilation, it was necessary to perform minor modifications to the template files and original grammar file used for the NPSNET Profile. Not surprisingly, the difficulties encountered during this process related soley to the issue of entity naming as discussed earlier in this work.

VII. FINDINGS AND FUTURE RESEARCH

The future of DIS will largely depend on its ability to adapt to the growing user demand for distributed simulation and to support the larger, more dynamic "virtual worlds" of tomorrow. In this context, the "scalability" of the protocol is one of the principle issues that must be addressed. The magnitude and complexity of this problem indicates that a solution may only be found through the collective effort of the entire DIS community. This thesis was prepared as part of that collective effort.

This chapter provides a summary of the findings and conclusions resulting from this thesis. The following information reflects the trails and tribulations experienced in formulating a DIS descriptive grammar, in developing the Protocol Support Utility, and in the application of both. Also presented below is a synopsis of related topics which may be considered for future investigation.

A. FINDINGS

As discussed earlier, this thesis effort was organized into three distinct phases. The findings and results of this work may be similarly organized.

First, those findings which coincide with efforts to **formulate a suitable DIS descriptive grammar**:

-- A modified BNF grammar was successfully used to describe DIS protocol entities. The attraction in using this approach to model the protocol was its inherent simplicity.

-- The ASN.1 syntax was considered to be overly complex for the purposes of this thesis. However, should a rigorous, more formal definition of the DIS protocol be required, ASN.1 should be considered.

Findings which coincided with the **application of the descriptive grammar to the DIS protocol**:

-- A simple grammar may be used to model the data elements associated with a complex protocol. The use of a modified BNF syntax proved effective in describing the

protocol entities defined in the current DIS standard. The resultant specification was both readable and suitable for parsing.

-- Several ambiguities were noted when attempting to model the DIS protocol. Generally, these ambiguities stemmed from lack of a clearly defined naming convention for individual protocol entities.

-- Though the modified BNF syntax was suitable for modeling the PDUs and data entities associated with DIS, it was not well suited for modeling the methods or implementation details needed in processing these entities. This deficiency became apparent when modeling the protocol, and was confirmed during development of the Protocol Support Utility.

In this regard, in may be necessary to forego the simplicity of the BNF grammar and explore a syntax better suited for modeling methods as well as data. Again, ASN.1 may prove to be the grammar of choice.

Finally, findings which coincided with the **development and implementation of the Protocol Support Utility**:

-- Similar to those encountered in modeling the protocol, implementation difficulties were experienced due to the lack of established naming conventions in DIS. Due to redundancies in naming, the current Enumeration Document [IST93] does not lend itself to automated code generation.

-- The DIS enumeration data should be published as a library or header file for direct implementation. Coincident with this effort, a standard naming convention should be adopted for all DIS protocol entities.

-- Without a clearly defined API, automated code generation is problematic. This finding has more to do with local implementation practices than it does the DIS protocol. A more generic network harness should be developed, one not dependent on specific protocol entity definitions. Once implemented, this network harness would serve as the API needed to accommodated automated source code production for DIS applications.

B. CONCLUSIONS

The objective of this thesis was to develop a method and mechanism to facilitate future DIS protocol improvement efforts. This objective was met through the formulation and application of a descriptive DIS grammar, and by developing and demonstrating the Protocol Support Utility. As a proof of concept, this thesis further demonstrated the possibility of using an automated tool to assist in DIS protocol modeling and applications development.

The ultimate value of this thesis effort will be determined by the extent to which this work contributes to future DIS protocol improvements. If the "scalability" of DIS is to be addressed by refining the form and content of DIS PDUs, then this work may well apply.

C. TOPICS FOR FURTHER RESEARCH

DIS is an ever evolving technology. Future research opportunities can be found in virtually every facet of DIS. In [IST94], the DIS community succinctly defined its objectives for future research. These objectives were discussed earlier in this work and need not be reiterated here. However, there does exist a number of topics which stem specifically from this work.

With regard to the Protocol Support Utility, the following may be explored:

-- Addition of a compliance validation feature. This feature would provide a means to verify that a new or refined protocol element is consistent with a given DIS standard. Addition of this feature would extend the utility of the tool to include protocol Verification, Validation and Accreditation (VV&A) as well as ongoing Configuration Management efforts.

-- Refinement of the protocol edit functions within the Protocol Support Utility. If desired, a template-based editor may be developed as an alternative to the use of **jot**.

-- Add code generation capabilities for other languages (if demand exists). Also, the refinement of the current Application Profiles and the addition of new Profiles might be pursued.

-- Experimentation with new or modified PDUs to measure deltas in host system performance.

Finally, additional topics which are not specifically related to this thesis but may hold merit as paths for future DIS research include:

-- Develop mechanism for real-time adaptation to new/modified PDUs.

-- Implementation of a more dynamic protocol paradigm. Candidates for consideration may include scripted or interpretive protocols (i.e. Telescript), introduction of migratory objects, or the use of self-describing protocol entities.

APPENDIX A: DIS PROTOCOL SPECIFICATION

This section provides a structured specification of the Protocol Data Units defined in the current DIS standard. The specific grammar used is a modified version of Backus-Naur Form (BNF) as described earlier in this report. Low level, bit-oriented data types are represented by a descriptive name followed by the number of bits in the field. For example, uint16 indicates a sixteen bit unsigned integer. Similarly, pad16 represents a sixteen bit data element used for byte alignment/padding.

Protocol Data Units (PDUs)

Entity Information/Interaction Protocol Family

<<EntityStatePDU>> ::=	<PDU_Header> <Entity_ID> Force_ID num_articulation_param <EntityType> <Alt_EntityType> <Entity_Linear_Velocity> <Location_Entity> <Entity_Orientation> entity_appearance <DR_Parameters> <Entity_Marking> capabilities <Articulation_Parameters>[num_articulation_param] ;
<<Collision_PDU>> ::=	<PDU_Header> <Issuing_Entity_ID> <Colliding_Entity_ID> <Event_ID> padding16 <Velocity> mass <Location> ;

Warfare Protocol Family

```
<<Fire_PDU>> ::=                <PDU_Header>
                                <Firing_Entity_ID>
                                <Target_Entity_ID>
                                <Munition_ID>
                                <Event_ID>
                                  padding32
                                <Launch_Location>
                                <Burst_Descriptor>
                                <Velocity>
                                  range
                                ;

<<Detonation_PDU>> ::=          <PDU_Header>
                                <Firing_Entity_ID>
                                <Target_Entity_ID>
                                <Munition_ID>
                                <Event_ID>
                                <Velocity>
                                <Location_Detonation>
                                <Burst_Descriptor>
                                <Location_Target>
                                  Detonation_Result
                                  num_articulation_param
                                  padding16
                                <Articulation_Parameters>[num_articulation_param]
                                ;
```

Logistics Protocol Family

<<**Service_Request_PDU**>> ::=
<PDU_Header>
<Requesting_Entity_ID>
<Servicing_Entity_ID>
Service_Type
num_supply_types
padding16
<Supplies> [num_supply_types]
;

<<**Resupply_Offer_PDU**>> ::=
<PDU_Header>
<Receiving_Entity_ID>
<Supplying_Entity_ID>
num_supply_types
padding24
<Supplies> [num_supply_types]
;

<<**Resupply_Receive_PDU**>> ::=
<PDU_Header>
<Receiving_Entity_ID>
<Supplying_Entity_ID>
num_supply_types
padding24
<Supplies> [num_supply_types]
;

<<**Resupply_Cancel_PDU**>> :: =
<PDU_Header>
<Receiving_Entity_ID>
<Supplying_Entity_ID>
;

<<**Repair_Complete_PDU**>> ::=
<PDU_Header>
<Receiving_Entity_ID>
<Repairing_Entity_ID>
RepairType
padding16
;

<<**Repair_Response_PDU**>> ::=
<PDU_Header>
<Receiving_Entity_ID>
<Repairing_Entity_ID>
Repair_Result
padding8[3]
;

Simulation Management Protocol Family

<<Create_Entity_PDU>> :=:

<PDU_Header>
<Originating_Entity_ID>
<Receiving_Entity_ID>
Request_ID
;

<<Remove_Entity_PDU>> ::=

<PDU_Header>
<Originating_Entity_ID>
<Receiving_Entity_ID>
Request_ID
;

<<Start-Resume_PDU>> ::=

<PDU_Header>
<Originating_Entity_ID>
<Receiving_Entity_ID>
<Real_World_Time>
<Simulation_Time>
Request_ID
;

<<Stop-Freeze PDU>> ::=

<PDU_Header>
<Originating_Entity_ID>
<Receiving_Entity_ID>
<Real_World_Time>
Reason
Frozen_Behavior
padding16
Request_ID
;

<<Acknowlegde_PDU>> ::=

<PDU_Header>
<Originating_Entity_ID>
<Receiving_Entity_ID>
Acknowledge_Flag
Response_Flag
Request_ID
;

<<Action_Request_PDU>> ::=

<PDU_Header>
<Originating_Entity_ID>
<Receiving_Entity_ID>
padding32
Request_ID
Action_ID
num_fixed_datum_rec
num_variable_datum_rec
<Fixed_Datum_Records> [num_fixed_datum_rec]
<Variable_Datum_Records>[num_variable_datum_rec]

```
                                   ;

<<Action_Response_PDU>> ::=        <PDU_Header>
                                   <Originating_Entity_ID>
                                   <Receiving_Entity_ID>
                                     padding32
                                     Request_ID
                                     Request_Status
                                     num_fixed_datum_rec
                                     num_variable_datum_rec
                                   <Fixed_Datum_Records> [num_fixed_datum_rec]
                                  <Variable_Datum_Records>[num_variable_datum_rec]
                                   ;

<<Data_Query_PDU>> ::=             <PDU_Header>
                                   <Originating_Entity_ID>
                                   <Receiving_Entity_ID>
                                     Request_ID
                                     time_interval
                                     num_fixed_datum_rec
                                     num_variable_datum_rec
                                   <Fixed_Datum_Records> [num_fixed_datum_rec]
                                  <Variable_Datum_Records> [num_variable_datum_rec]
                                   ;

<<Set_Data_PDU>> ::=               <PDU_Header>
                                   <Originating_Entity_ID>
                                   <Receiving_Entity_ID>
                                     Request_ID
                                     padding32
                                     num_fixed_datum_rec
                                     num_variable_datum_rec
                                   <Fixed_Datum_Records> [num_fixed_datum_rec]
                                  <Variable_Datum_Records>[num_variable_datum_rec]
                                   ;

<<Data_PDU>> ::=                   <PDU_Header>
                                   <Originating_Entity_ID>
                                   <Receiving_Entity_ID>
                                     Request_ID
                                     padding32
                                     num_fixed_datum_rec
                                     num_variable_datum_rec
                                   <Fixed_Datum_Records> [num_fixed_datum_rec]
                                  <Variable_Datum_Records>[num_variable_datum_rec]
                                   ;
```

```
<<Event_Report_PDU>> ::=        <PDU_Header>
                                <Originating_Entity_ID>
                                <Receiving_Entity_ID>
                                  Event_Type
                                  padding32
                                  num_fixed_datum_rec
                                  num_variable_datum_rec
                                <Fixed_Datum_Records>[num_fixed_datum_rec]
                                <Variable_Datum_Records>[num_variable_datum_rec]
                                ;

<<Message_PDU>> ::=             <PDU_Header>
                                <Originating_Entity_ID>
                                <Receiving_Entity_ID>
                                  padding32
                                  num_fixed_datum_rec
                                  num_variable_datum_rec
                                <Fixed_Datum_Records>[num_fixed_datum_rec]
                                <Variable_Datum_Records>[num_variable_datum_rec]
                                ;
```

Distributed Emission Regeneration Family

```
<<Electromag_Emission_PDU>> ::=   <PDU_Header>
                                  <Emitting_Entity_ID>
                                  <Event_ID>
                                    state_update_indicator
                                    num_systems
                                    padding16
                                  <Emitting_System_Data>[num_systems]
                                  ;

<<Designator_PDU>> ::=            <PDU_Header>
                                  <Designating_Entity_ID>
                                    Code_Name
                                  <Designated_Entity_ID>
                                    padding8
                                    designator_code
                                    designator_power
                                    Designator_Wavelength
                                  <Designator_Spot>
                                  <Designator_Spot_Loc>
                                  ;

<<Transmitter_PDU>> ::=           <PDU_Header>
                                  <Entity_ID>
                                    Radio_ID
                                  <Radio_Entity_Type>
                                    Transmit_State
                                    Input_Source
                                    padding16
                                  <Antenna_Location>
                                  <Relative_Antenna_Location>
                                    Antenna_Pattern_Type
                                    antenna_pattern_length
                                    Frequency
                                    Transmit_Frequency_Band
                                    Power
                                  <Modulation_Type>
                                    Crypto_System
                                    Crypto_Key_ID
                                    length_modulation_param
                                    padding24
                                  <Modulation_Param> [length_modulation_param]
                                  <Antenna_Pattern_Param> [antenna_pattern_length]
                                  ;
```

<<Signal_PDU>> ::=	<PDU_Header> <Entity_ID> Radio_ID Encoding_Scheme TDL_Type Sample_Rate data_length Samples <Signal_Data> [data_length] ;
<<Receiver_PDU>> ::=	<PDU_Header> <Entity_ID> Radio_ID Receiver_State padding16 Received_Power <Transmitter_Entity_ID> Transmitter_Radio_ID ;

Composite Data Types

<Alt_Entity_Type> ::=
Entity_Kind
Domain
Country
Category
SubCategory
Specific
Extra
;

<Angular_Vel_Vector> ::=
rate_about_x
rate_about_y
rate_about_z
;

<Antenna_Location> ::=
<World_Coord>
;

<Antenna_Pattern_Param> ::=
<Omni_Pattern> |
<Beam_Pattern> |
<Spherical_Pattern>
;

<Articulation_Parameters> ::=
P_Type_Designator
change_indicator
ID-Part_Attached_to
parameter_type
parameter_value[8]
;

<Beam_Data> ::=
beam_data_length
Beam_ID
beam_param_index
<Fund_Param_Data>
Feam_Function
num_targets
HD_Track_Jam
padding8
Jam_Mode_Seq
<Track_Jam>
;

<Burst_Descriptor> ::=
<Munition>
Warhead
Fuse
quantity
rate
;

<Beam_Direction> ::=
psi
theta
phi
;

<Beam_Pattern> ::=
<Beam_Direction>
beam_az
beam_elev
Reference_Sys
padding24
Ez
Ex
Phase
;

<Colliding_Entity_ID> ::=
<Entity_ID>
;

<Designated_Entity_ID> ::=
<Entity_ID>
;

<Designating_Entity_ID> ::=
<Entity_ID>
;

<Designator_Spot> ::=
<Entity_Coord>
;

<Designator_Spot_Loc> ::=
<World_Coord>
;

<DR_Param> ::=
DR_Algorithm
Other_DR_Parameters
<Linear_Vel_Vector>
<Angular_Vel_Vector>
;

<Emitter_System> ::=
Emitter_Name
Emitter_Function
Emitter_ID
;

<Emitting_Entity_ID> ::=
<Entity_ID>
;

<Emitting_System_Data> ::=
system_data_length
num_beams
padding16
<Emitter_System>
<Location_Emitter>
<Beam_Data>
;

<Entity_Coord> ::=	x_component y_component z_component ;
<Entity_ID> ::=	**<Sim_Address>** Entity_ID ;
<Entity_Linear_Velocity> ::=	x_component y_component z_component ;
<Entity_Marking> ::=	Character_Set markings[11] ;
<Entity_Orientation> ::=	**<Euler_Angle>** ;
<Entity_Type> ::=	Entity_Kind Domain Country Category SubCategory Specific Extra ;
<Euler_Angle> ::=	psi theta phi ;
<Event_ID> ::=	**<Entity_ID>** ;
<Firing_Entity_ID> ::=	**<Entity_ID>** ;
<Fixed_Datum_Records> ::=	Fixed_Datum_ID Fixed_Datum ;

<Fund_Param_Data> ::=
beam_frequency
beam_freq_range
erp
prf
pulse_width
beam_az_center
beam_az_sweep
beam_elev_center
beam_elev_sweep
beam_sweep_sync
;

<Issuing_Entity_ID> ::= **<Entity_ID>**
;

<Linear_Vel_Vector> ::=
vector_component1
vector_component2
vector_component3
;

<Location_Emitter> ::= **<Entity_Coord>**
;

<Location_Entity> ::= **<World_Coord>**
;

<Location_Launch> ::= **<Entity_Coord>**
;

<Location_Detonation> ::= **<World_Coord>**
;

<Location_Target> ::= **<Entity_Coord>**
;

<Modulation_Type> ::=
Spread_Spectrum
Major
Detail
System
;

<Modulation_Param> ::= uint8
;

<Munition> ::= **<Entity_Type>**
;

<Munition_ID> ::= **<Entity_ID>**
;

<Originating_Entity_ID> ::= **<Entity_ID>**

```
                                        ;

<PDU_Header> ::=                          protocol_version
                                          exercise_id
                                          PDU_Type
                                          protocol_family
                                          time_stamp
                                          length
                                          padding16
                                        ;

<Radio_Entity_Type> ::=                 <Entity_Type>
                                        ;

<Real_World_Time> ::=                     hour
                                          time_past_hour
                                        ;

<Receiving_Entity_ID> ::=               <Entity_ID>
                                        ;

<Relative_Antenna_Location> ::=         <Entity_Coord>
                                        ;

<Repairing_Entity_ID> ::=               <Entity_ID>
                                        ;

<Requesting_Entity_ID> ::=              <Entity_ID>
                                        ;

<Servicing_Entity_ID> ::=               <Entity_ID>
                                        ;

<Signal_Data> ::=                         digit_audio
                                        ;

<Sim_Address> ::=                         Site_ID
                                          Application_ID
                                        ;

<Simulation_Time> ::=                     hour
                                          time_past_hour
                                        ;

<Supplies> ::=                          <Supply_Type>
                                        ;

<Supplying_Entity_ID> ::=               <Entity_ID>
                                        ;
```

<Supply_Type> ::=	**<Entity_Type>** quantity_supply ;
<Target_Entity_ID> ::=	**<Entity_ID>** ;
<Transmitter_Entity_ID> ::=	**<Entity_ID>** ;
<Track_Jam> ::=	**<Entity_ID>** Emitter_ID Beam_ID ;
<Variable_Datum_Records> ::=	Variable_Datum_ID variable_datum_length variable_datum_value {padding} ;
<Velocity> ::=	x_component y_component z_component ;
<World_Coord> ::=	x_coord y_coord z_coord ;

Atomic Data Types

acknowledge_flag ::=	enum16;
Action_ID ::=	uint32;
antenna_pattern_length::=	uint16;
Antenna_Pattern_Type ::=	enum16;
Application ::=	uint16;
Application_ID ::=	uint16;
beam_az ::=	float32;
beam_az_center ::=	float32;
beam_az_sweep ::=	float32;
beam_elev ::=	float32;
beam_elev_center ::=	float32;
beam_elev_sweep ::=	float32;
beam_data_length ::=	uint8;
beam_frequency ::=	float32;
beam_freq_range ::=	float32;
Beam_Function ::=	enum8;
Beam_ID ::=	uint8;
beam_param_index ::=	uint16;
beam_sweep_sync ::=	float32;
capabilities ::=	bool32;
Category ::=	enum8;
change_indicator ::=	uint8;
Character_Set ::=	enum8;
Code_Name ::=	enum16;
Country ::=	enum16;
Crypto_Key_ID ::=	uint16;

Crypto_System ::=	enum16;
data_length ::=	int16;
designator_code ::=	enum8;
designator_power ::=	float32;
Designator_Wavelength ::=	float32;
Detail ::=	enum16;
Detonation_Result ::=	enum8;
digit_audio ::=	uint8;
Domain ::=	enum8;
DR_Algorithm ::=	enum8;
Emitter_Function ::=	enum8;
Emitter_ID ::=	uint8;
Emitter_Name ::=	enum16;
Encoding_Scheme ::=	enum16;
antity_appearance ::=	enum32;
Entity_ID ::=	uint16;
Entity_Kind ::=	enum8;
ERP ::=	float32;
event ::=	uint16;
Event_Type ::=	enum32;
Exercise_ID ::=	uint8;
ex ::=	float32;
Extra ::=	enum8;
ez ::=	float32;
Fixed_Datum ::=	enum32;

Fixed_Datum_ID ::=	uint32;
Force_ID ::=	enum8;
Frequency ::=	uint64;
Frozen_Behavior ::=	enum8;
Fuse ::=	enum16;
HD_Track_Jam ::=	enum8;
hour ::=	int32;
ID-Part_Attached_to ::=	uint16;
Input_Source ::=	enum8;
Jam_Mode_Seq ::=	uint32;
length ::=	uint16;
length_modulation_param ::=	uint8;
markings ::=	uint8;
Major ::=	enum16;
mass ::=	float32;
num_articulation_param ::=	uint8;
num_beams ::=	uint8;
num_fixed_datum_rec ::=	uint32;
num_supply_types ::=	uint8;
num_systems ::=	uint8;
num_targets ::=	uint8;
num_variable_datum_rec ::=	uint32;
Other_DR_Param ::=	enum120;
padding ::=	pad8;
padding8 ::=	pad8;
padding16 ::=	pad16;

padding24 ::=	pad24;
padding32 ::=	pad32;
parameter_value ::=	int64;
param_type ::=	enum32;
Phase ::=	float32;
P_Type_Designator ::=	enum8;
PDU_Type ::=	enum8;
phi ::=	float32;
Power ::=	float32;
prf ::=	float32;
protocol_family ::=	enum8;
protocol_version ::=	enum8;
psi ::=	float32;
Pulse_Width ::=	float32;
quantity ::=	uint16;
quantity_supply ::=	float32;
Radio_ID ::=	uint16;
range ::=	float32;
rate ::=	uint16;
rate_about_x ::=	float32;
Rrte_about_y ::=	float32;
rate_about_z ::=	float32;
Reason ::=	enum8;
Receiver_Power ::=	float32;
Receiver_State ::=	enum16;

Reference_Sys ::=	enum8;
Repair ::=	enum16;
Repair_Result ::=	enum8;
Request_ID ::=	uint32;
Request_Status ::=	enum32;
Response_Flag ::=	enum16;
Samples ::=	int16;
Sample_Rate ::=	int32;
Service_Type ::=	enum8;
site ::=	uint16;
Site_ID ::=	uint16;
Specific ::=	enum8;
Spread_Spectrum ::=	enum16;
State_Update_Indicator ::=	enum8;
SubCategory ::=	enum8;
System ::=	enum16;
system_data_length ::=	uint8;
TDL_Type ::=	enum16;
theta ::=	float32;
time_interval ::=	uint32;
time_past_hour ::=	uint32;
time_stamp ::=	uint32;
Transmit_Frequency_Band ::=	float32;
Transmit_State ::=	enum8;
Transmitter_Radio_ID	uint16;
variable_datum_length ::=	uint32;

Variable_Datum_ID ::=	enum32;
variable_datum_value ::=	enum32;
vector_component1 ::=	float32;
vector_component2 ::=	float32;
vector_component3 ::=	float32;
Warhead ::=	enum16;
x_component ::=	float32;
x_coord ::=	float64;
y_component ::=	float32;
y_coord ::=	float64;
z_component ::=	float32;
z_coord ::=	float64;

APPENDIX B: DIS LEX SOURCE LISTING

The source code provided below is a LEX-based analyzer used to process a DIS protocol grammar. The accepted grammar is the same modified BNF syntax shown in Appendix A.

```
Program:    DIS LEX

Written by: Mike Canterbury
Date:         950708

Purpose:    This function serves as a lexical analyzer
            used to parse a DIS grammar specification.
            The function is implemented using LEX syntax
            and  is part of the Protocol Support Utility.
```

```
%s C_CMMT CPP_CMMT PDU_DEF STRUCT_DEF
%{

#include <string.h>

#ifndef PDU_SUPPORT_GLOBALS
#include "genCLASS_globals.h"
#endif

#ifndef PDU_SUPPORT_FUNCTIONS
#include "genCLASS_funcs.h"
#endif

char tokenString[MAX_STRING];
char structString[MAX_STRING];
char dataString[MAX_STRING];

int  LHS = 1;


/*   The below function strips the special delimiter symbols  */
/*  from the scanned tokens.                                  */
/*                                                           */
void stripToken(char* inString,int tokenType, int yyleng)
  {
  int  count;
  char *tempString = "";

  for(count=tokenType;count<(yyleng-tokenType);count++)
    {
    if (inString[count] != ' ')
      tempString[count-tokenType] = inString[count];
    }
  tempString[count-tokenType] = '\0';
  strcpy(tokenString,tempString);
  }
%}
```

```
%%

[ \t]*                  { /*  skip whitespace  */ };
<INITIAL>[ \n]*|
<PDU_DEF>[ \n]*|
<STRUCT_DEF>[ \n]*      { /*  end of line  */ };

"/*"                    { BEGIN C_CMMT; };
<C_CMMT>[^*/]           ;
<C_CMMT>"*/"            { BEGIN INITIAL; };

"//"                    { BEGIN CPP_CMMT; };
<CPP_CMMT>[^\n]*        ;
<CPP_CMMT>[\n] { BEGIN INITIAL; };

<INITIAL>\<\<[A-Za-z_][-A-Za-z0-9_]*\>\>
                              { BEGIN PDU_DEF;
                                stripToken(yytext,2,yyleng);
                                 addSymbolPDU(tokenString);
                              };

<INITIAL>\<[A-Za-z_][-A-Za-z0-9_]*\>
                              { BEGIN STRUCT_DEF;
                                 stripToken(yytext,1,yyleng);
                                 addSymbolCOMP(tokenString, LEFT, NONE);
                              };

\:\:\=                           {LHS = 0; };


<INITIAL>[-A-Za-z0-9_]*
                              { stripToken(yytext,0,yyleng);
                                if (LHS)
                                    {
                                    addSymbolATOM(tokenString, LEFT, NONE);
                                    }
                                else
                                   {
                                     addSymbolATOM(tokenString, RIGHT, NONE);
                                   };
                              };

<PDU_DEF>\<[A-Za-z_][-A-Za-z0-9_]*\>        |
<STRUCT_DEF>\<[A-Za-z_][-A-Za-z0-9_]*\>
                              { stripToken(yytext,1,yyleng);
                                  addSymbolCOMP(tokenString, RIGHT, NONE);
                              };
<PDU_DEF>\{\<[A-Za-z_][-A-Za-z0-9_]*\>\}      |
<STRUCT_DEF>\{\<[A-Za-z_][-A-Za-z0-9_]*\>\}
                              { stripToken(yytext,2,yyleng);
                                  addSymbolCOMP(tokenString, RIGHT, LINK);
                              };
<PDU_DEF>\[[-A-Za-z0-9_*+/]*\]
                                { stripToken(yytext,0,yyleng);
                                  addPDU_Array(tokenString);
                              };
<STRUCT_DEF>\[[-A-Za-z0-9_*+/]*\]
                              { stripToken(yytext,0,yyleng);
                                  addCOMP_Array(tokenString);
                              };


<PDU_DEF>[-A-Za-z0-9_]*                      |
<STRUCT_DEF>[-A-Za-z0-9_]*
                                { stripToken(yytext,0,yyleng);
                                  addSymbolATOM(tokenString, RIGHT, NONE);
                                };
```

```
<INITIAL>\;                        {
                               LHS = 1;
                         objectCOMPLETE(ATOM_DEF_STATE);
                               };
<PDU_DEF>\;                        {
                                BEGIN INITIAL;
                                LHS = 1;
                                  objectCOMPLETE(PDU_DEF_STATE);
                                  };
<STRUCT_DEF>\;                    {
                                  BEGIN INITIAL;
                               LHS = 1;
                                  objectCOMPLETE(COMP_DEF_STATE);
                               };

%%


int yywrap()         /*  close the input file   */
  {
  fclose(yyin);
  return 1;
  }
```

APPENDIX C: PROTOCOL SUPPORT UTILITY USER'S GUIDE

The Protocol Support Utility is intended as a tool to be used in the development and refinement of DIS data elements (PDUs, composite data types, atomic data types). The following describes the installation, operation, and use of this tool.

System Hardware Requirements

The Protocol Support Utility was developed for use on platforms which support typical UNIX-hosted, X-Window environments. Beyond the need for minimal runtime disk space, no other special resources or hardware facilities are required.

Installation

The following files are required for proper program operation. All files should be installed in the same directory in which the Utility is to be used.

File	Use
disPSU	Protocol Support Utility
_defaultPDU.gram	Default/initialization grammar

The following files are required to support source code generation for the Application Profiles specified:

NPSNET Profile

__enum.h	Enumeration data file (source:NPSNET pdu.h)
__disbridge.cc	Source code template (DIS_bridge)
__disnetlib.cc	Source code template (DIS_net_manager)
__disbridge_depend.cc	Implementation dependent source
__disnetlib_depend.cc	Implementation dependent source

Class-based Profile

__enum.h	Enumeration data file (source:NPSNET pdu.h)
__disbridgeCLASS.cc	Class template (DIS_bridge)
__disnetlibCLASS.cc	Class template (DIS_net_manager)

Silicon Graphics provides an ASCII-based text editor, **jot**, for default use on most SGI workstations. The Protocol Support Utility uses this program as the principle means of editing the protocol grammar. To ensure proper operation of this feature, **jot** should be installed in the following location:

/usr/sbin/jot

This is the default installation path for **jot** on most systems. If **jot** is not available, or another editor is preferred, an alternate program may be used. However, the use of an alternate editor requires that minor changes be made to the Protocol Support Utility, and the entire system recompiled. Appendix D discusses the specific source code changes required.

Program Initiation

The Protocol Support Utility may be initiated from the UNIX command line as follows:

% disPSU

If desired, an initial grammar file may be loaded on program start-up by providing the name of the desired grammar file when invoking the Utility as follows:

% disPSU ***myGrammar.gram***

User Interface

As can be seen in Figure C-1, the user interface provided with the Protocol Support Utility consists of a Main Program Window which supports a pulldown menu bar, a display window for runtime status messages, and a viewing window in which working grammar definitions may be displayed. Also provided are buttons to initiate the editing and update of specific grammar constructs.

The pulldown menu bar has two options, **File** and **Generate**. The **File** menu is used to manage the grammar files processed by the Utility, while the **Generate** option is used to initiate source code production as appropriate. The particular facilities provided under each option are discussed below.

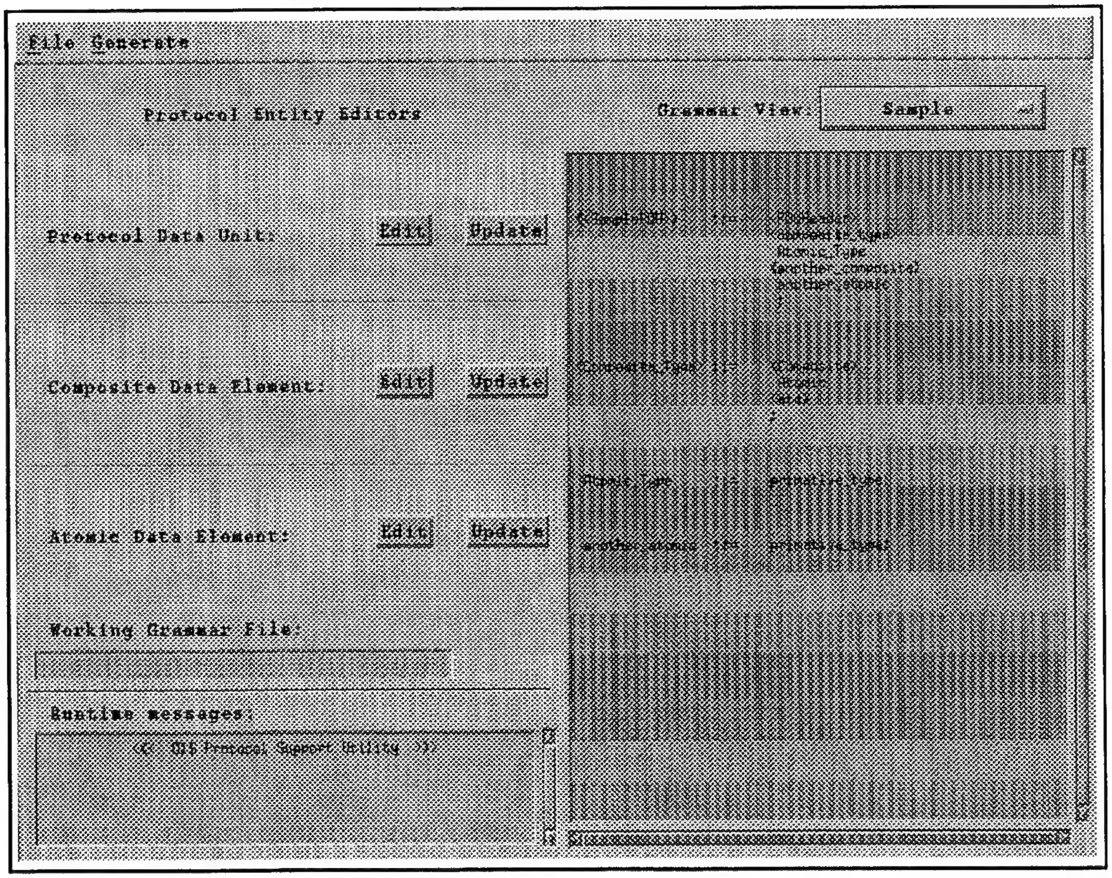

Figure C-1. Protocol Support Utility User Interface

The **Grammar View** window provides a facility to view the grammar-based representation of DIS protocol elements. This is window is for display only and editing is not supported. The contents of the viewing window may be selected by means of the pulldown widget labeled "**Grammar View**". This widget is located directly above the viewing window and offers the following selections:

-- *Sample* - allows viewing of a sample grammar specification.

-- *PDU* - allows viewing of all PDUs defined in the current working file.

-- *Composite* - allows viewing of all Composite types defined.

-- *Atomic* - allows viewing of all Atomic types defined.

-- *All* - allows viewing of all PDUs, Composite, and Atomic types defined.

The **Grammar View** window may be scrolled to view protocol definitions which have been selected for viewing but do not appear within the text window. If necessary, the window may be refreshed by reselecting the particular grammar view desired.

The **Runtime Message** window is located in the lower portion of the Main Window. This purpose of this window is to display the program's runtime status and error messages as they occur. Like the **Grammar View** display, this window may be scrolled to view previous messages which do not appear within the current viewing area.

Located immediately above the **Runtime Message** window is the **Working Grammar File** window. This single line, text window displays the path and filename of the grammar file currently active in the Utility. Like the **Grammar View** and **Runtime Message** windows, the **Working Grammar File** window is used for the display of information only and may not be edited.

Grammar Files

The **File** option on the Menu Bar provides facilities to **Open** and **Merge** pre-existing grammar files, as well as created **New** files for protocol development work. It should be noted that any grammar file loaded must be in a format consistent with the DIS descriptive grammar specified for use with this Utility. A brief discussion of the grammar is presented later in this guide.

Also found under the **File** option are facilities to **Save** the working grammar file, **Close** the working grammar file, and **Exit** the program.

Authoring and Editing

This Utility supports the use of a simple grammar to describe DIS protocol entities. The Grammar has three basic constructs:

<< ProtocolDataUnits >>
< CompositeDataTypes >
AtomicDataTypes

These constructs are used to model the individual protocol entities associated with DIS. The following example illustrates the use of this grammar in describing a typical DIS entity:

<<SamplePDU>> ::= **<PDUHeader>**
<EntityID>
ForceID
params
;

Note the difference in the representation of the two Atomic data elements, **ForceID** and **params**. The capitalization of letters in the first element is significant. This is a naming convention that may be used within an Applications Profile to distinguish between Atomic elements which are to be defined as data types and those which are merely variables of some base data type supported by a given programming language.

Naming conventions are an important consideration when describing protocol entities. Care should be taken that the names given to protocol entities be consistent with those used in the system for which code is to be generated. Failure to do so will result in compilation problems when the generated code is integrated into the DIS host system.

DIS descriptive grammars may be authored or edited using any ASCII-based text editor. As mentioned above, the editing facility incorporated into the Protocol Support Utility is **jot**, a SGI-supplied product. Grammar edit functions are initiated by depressing the **Edit** button associated with the particular DIS protocol construct to be edited. When a particular edit function is invoked, a **jot** session is launched. When invoked, the **jot** editor will contain the grammar description of the DIS entities associated with the current session. For example, the PDU Editor may be launched by depressing the Protocol Data Unit **Edit** button. This done, a **jot** session will appear and the session window will contain all PDUs currently defined in the Utility's internal PDU table.

The grammar displayed in any **jot** session may be edited as desired. The filename reflected by the jot editor (seen above the jot window) is a scratch file created by the Utility. Once editing is complete, this file must be saved if changes to the grammar are to be made.

Once the altered grammar file has been saved, the **jot** editor may be exited. To effect any changes made during an edit session, the corresponding **Update** button must be depressed. Initiating the **Update** function will effectively reinitialize the internal symbol tables within the Utility, thereby incorporating any changes made to the grammar. There is no "undo" feature currently implemented in the Utility.

Source Code Generation

Code Generation is an automated process. Any **working grammar** may be used to generate DIS program source code. A **working grammar** is the grammar definition in use by the program at the time that Code Generation is initiated. As mentioned earlier, the file associated with the **working grammar** is displayed in the **Working Grammar File** window.

To initiate **Code Generation**, select the appropriate Applications Profile under the **Generate** menu located on the Main Menu Bar. Currently, two Applications Profiles have been implemented. The first, NPSNET Profile, generates source code based upon NPSNET IV [Mace94]. The second profile, Class-based Profile, is intended as an example of C++ source code production. The source code generated by this profile should not be considered to be complete. It does, however, provide a starting point from which an object- oriented network harness might be produced.

Additional Profiles may be added to the Utility if desired. Similarly, the current Profiles may be modified or improved if required. Appendix D discusses the program modifications necessary to incorporate this type of change.

Program Termination

The Protocol Support Utility may be terminated by selecting the **Exit** option in the **File** menu located on the Main Menu Bar or by closing ("double-clicking") the main program window. Care should be taken to save all work prior to program termination.

APPENDIX D: APPLICATION SUPPORT INFORMATION

The viability of the Protocol Support Utility may be dictated by the ease in which it can be maintained, modified, and enhanced. The following information is provided to assist in future efforts to support this application, whether intended to improve the tool or to correct deficiencies which may be discovered through continued use.

Files

The following files constitute the source code necessary to maintain and regenerate the Protocol Support Utility.

File	Use
pduGUI.C	Source Code - User Interface
pduGUI.h	Header - User Interface
disPSU_GLOBAL_DEFINES.h	Global definitions
genCLASS.C	Source Code - LEX Interface/Table Management
genCLASS_globals.h	Header - Global type declarations
genCLASS_funcs.h	Header - Function prototypes
genLEX.l	Source Code - Scanner/Lexical Analysis
lex.yy.c	Source Code - LEX generated source file
genLEX.h	Header - LEX function prototype
_classPROFILE.C	Source Code - Class-based Application Profile
_classPROFILE.h	Header - Application profile prototypes
_npsPROFILE.C	Source Code - NPSNET Applications Profile
_npsPROFILE.h	Header - Application profile prototypes
Makefile	Program Build

In addition to the files noted above, the following are required for program execution:

disPSU	Protocol Support Utility
_defaultPDU.gram	Default/initialization grammar

_NPSgram.gram	Representative NPSNET grammar
__enum.h	Enumeration data file (source:NPSNET pdu.h)
__disbridge.cc	Source code template (DIS_bridge)
__disnetlib.cc	Source code template (DIS_net_manager)
__disbridge_depend.cc	Implementation dependent source
__disnetlib_depend.cc	Implementation dependent source
__disbridgeCLASS.cc	Class template (DIS_bridge)
__disnetlibCLASS.cc	Class template (DIS_net_manager)

Default Environment

The default environment parameters for the Protocol Support Utility are provided in the global header file, **disPSU_GLOBAL_DEFINES.h**. The definitions established in this file include default filenames for the various grammar files used by the tool, as well as the specific path needed to invoke the default editor, **jot**. Also included in this header file are the definitions which specify internal table sizes and the maximum number of discrete symbols supported by this Utility. As presently implemented, these declarations include:

Declaration		Meaning
--- MAX_PDUS	50	PDU Table size, maximum number of PDUs allowed.
--- MAX_TYPES	200	Composite/Atomic Table sizes.
--- MAX_ELEMENTS	75	Number of elements which may comprise a PDU/ Composite type.
--- MAX_SYMBOLS	1000	Symbol Table size, maximum number of symbols.
--- MAX_STRING	30	Length of symbol (PDU, Composite, Atomic) name.

Editor

As noted in the paragraph above, the Protocol Support Utility uses **jot** as the principle means of editing the protocol grammar. This editor is initiated by means of a UNIX system call which passes the path necessary to invoke **jot** as well as the name of the scratch file to

be opened for use in the grammar edit process. A declaration within the global header file reflects the following path to the editor:

/usr/sbin/jot

This is the default installation path for **jot** on most systems. If **jot** is not available, or another editor is preferred, an alternate program may be used. This change may be made by simply modifying the path and editor portions of the following definitions (found in the global header file):

--- **PDU_EDITOR_PATH** **"/usr/sbin/jot _pduSCRATCH.GRAM"**
--- **COMP_EDITOR_PATH** **"/usr/sbin/jot _compSCRATCH.GRAM"**
--- **ATOM_EDITOR_PATH** **"/usr/sbin/jot _atomSCRATCH.GRAM"**

Lexical Analysis

Grammar files read by the Utility are processed by means of a LEX-based scanner. The scanner has been designed to parse tokens which signify DIS protocol entities (<<PDUs>>, <Composites>, Atomics) and is used repeatedly for the initialization and loading of all internal tables. The LEX syntax for the scanner may be found in **genLEX.l** and the same LEX source listing is included in Appendix B.

The LEX utility is used to generate the "C" language source code necessary to produce a scanner for a given grammar. In the case of the Protocol Support Utility, the LEX source included in **genLEX.l** is used for parsing DIS constructs. Changes to the DIS grammar will necessitate that corresponding changes be made to the LEX source.

Subsequent to any changes to the LEX source, the "C" language source code for the scanner must be regenerated by means of the LEX utility. This process may be accomplished at the UNIX command line by:

% lex genLEX.l

The result of using the LEX utility is a single "C" language source file, **yy.lex.c**. This file must be recompiled into the Protocol Support Utility to implement any changes to the DIS grammar. This may be simply accomplished by:

```
% make
```

Code Generation

Code Generation is an automated process. The Protocol Support Utility uses the concept of an Applications Profile as a means of producing the desired source code output. In essence, Code Generation is merely a process of extracting grammar information from the Symbol Tables within the program and translating this grammar into the language and programming constructs specified by the Applications Profile.

Currently, two Applications Profiles have been included with the Utility. The first, NPSNET, is based upon the network harness currently implemented in NPSNET IV [Mace94]. The files which constitute the functions and data used in this profile include:

_npsPROFILE.C	- Source Code Generation
_npsPROFILE.h	- Function prototypes
_NPSgram.gram	- NPSNET grammar
__enum.h	- Enumeration data file
__disbridge.cc	- Source code template (DIS_bridge)
__disnetlib.cc	- Source code template (DIS_net_manager)
__disbridge_depend.cc	- Implementation dependent source
__disnetlib_depend.cc	- Implementation dependent source

The NPSNET profile was built by identifying the specific functions within the network harness which were dependent upon the definition of protocol entities, whether PDUs or protocol data types. The functions which had little or no dependence on the protocol definition were placed in the source code template files listed above. The remaining protocol-dependent functions were analyzed to determine whether they were

implementation dependent or could be regenerated by the Protocol Support Utility. The implementation dependent functions were placed in the so named source files listed above. Those functions which were appropriate for automated source code production were generated by the utility as "C"-language constructs. The specific functions responsible for source code generation (under the NPSNET profile) can be found in **_npsPROFILE.C**. The actual source code generated by the Utility under the NPSNET Profile is a combination of the original NPSNET source and the code generated by the Utility.

The second, Class-based Profile is meant to demonstrate the ability of the Utility to generate source code based upon different programming constructs or paradigms. The same NPSNET network harness was used as a basis for this profile. However, this profile was intended as a demonstration in producing object-oriented source code and not as a means to generate an entire Class-based network harness. The files which constitute this profile include:

_classPROFILE.C	- Source Code generation
_classPROFILE.h	- Function prototypes
_NPSgram.gram	- NPSNET grammar
__enum.h	- Enumeration data file
__disbridgeCLASS.cc	- Class template (DIS_bridge)
__disnetlibCLASS.cc	- Class template (DIS_net_manager)

It is recommended that new profiles be built in a manner similar to that used on the two profiles discussed above. The functions necessary to extract Symbol Table data and to produce source code in the form and language of choice should be placed in a single ".C" file. In addition to any **include** files needed for the Profile in question, the following statements should be placed at the beginning of the ".C" file:

```
#include "disPSU_GLOBAL_DEFINES.h"
#ifndef GUI_SUPPORT_FUNCTIONS
#define GUI_SUPPORT_FUNCTIONS
#include "pduGUI.h"
```

```
#endif
#include "genCLASS_globals.h"
#include "_npsPROFILE.h"

extern SYMBOLS  symbolTABLE[MAX_SYMBOLS];
extern pduSTRUCT pduTABLE[MAX_PDUS];
extern compSTRUCT compositeTABLE[MAX_TYPES];
extern atomSTRUCT  atomicTABLE[MAX_TYPES];
extern int pduCOUNT;
extern int compositeCOUNT;
extern int atomicCOUNT;
```

An accompanying header file should be prepared which contains the prototypes for the functions associated with the Profile as well as any target filenames needed for program output. For example, the header file associated with the NPSNET profile (**_npsPROFILE.h**) includes prototypes for each function used to build the generated source code. Additionally, the header file specifies the following output files which will contain the code generated by the Utility:

```
#define NPSNET_PDU_HEADER          "NPSpdu.h"
#define NPSNET_DIS_BRIDGE_FILE  "NPSdisbridge.cc"
#define NPSNET_DIS_MANAGE_FILE  "NPSdisnetlib.cc"
```

To implement a new Profile, several minor changes will be required to the Protocol Support Utility. All necessary changes may be made in the **pduGUI.C** file.

First, it will be necessary to add a pull-down entry to the **Generate** menu. This can be done by adding the following source code to the **make_generate_menu()** function in the **pduGUI.C** file. Note: The name of the new Profile and callback function should be added as appropriate.

```
make_pulldown_entry(menupane, "new Profile", genNewCB,  (XtPointer) NULL);
```

Next, a callback function should added which invokes source code generation under the new Profile. The following is an example of the callback used to initiate source code production for the NPSNET Profile:

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* genNPSCB -                                                             */
/*  This function initiates generation of  the NPSNET Profile             */
/*  Source Code definitions.                                              */
void genNPSCB (Widget, XtPointer, XtPointer) {
   msgPrint(message_box ,">>> Generating NPSNET Profile Source...\n"
                              ,MESSAGE_WINDOW);
  npsnetCodeGen();
  } /* End genNPSCB... */
```

Finally, changes should be made to the **Makefile** to reflect the ".C" file and header file associated with the new Profile. The specific portion of the **Makefile** which must be modified is shown below. As an example, the entries necessary to incorporate the NPSNET Profile are depicted in bold type.

```
pduGUI:  pduGUI.o genCLASS.o genCLASS_funcs.h genLEX.h pduGUI.h \
         genCLASS_globals.h _npsPROFILE.h _classPROFILE.h disPSU_GLOBAL_DEFINES.h
         CC -o disPSU pduGUI.o lex.yy.o genCLASS.o _npsPROFILE.C _classPROFILE.C \
         $(CFLAGS) $(LIBS)
```

Enumeration Data

In the profiles above, DIS enumeration data is appended to the end of the protocol header file produced by the Utility (e.g **NPSpdu.h**). The enumeration data used in the current Profiles is contained in **_enum.h**. The data in this file is used as known to be outdated, but is consistent with that used in the most recent release of NPSNET. Up-to-date enumeration data can be found in [IST95a].

LIST OF REFERENCES

[Bate94] Bate, Gordon, Network Performance Simulation for Distributed Interactive Simulations, Proceedings of the 11th Workshop on Standards for the Interoperability for Defense Simulations, Institute of Simulation and Training, Orlando, Florida, September, 1994.

[Budd94] Budd, Timothy, A., *Classic Data Structures in C++*, Addison-Wesley Publishing Company, Menlo Park, California, 1994.

[Calv94] Calvin, James, O., Van Hook, Daniel J. *AGENTS: An Architectural Construct to Support Distributed Simulation,* Proceedings of the 11th Workshop on Standards for the Interoperability for Defense Simulations, Institute of Simulation and Training, Orlando, Florida, September, 1994.

[Case90] Case, Jeffery, D., *Panel Review of Long-Haul Networking in Distributed Simulations*, Institute of Defense Analysis Document D-780, January,1990.

[Dick94] Dickens, Alan R., *Self-Describing Entities in the Distributed Interactive Simulation Environment*, Proceedings of the 10thWorkshop on Standards for the Interoperability of Defense Simulations, Institute of Simulation and Training, Orlando, Florida, March, 1994.

[DSBR93] Defense Science Board Report, *Impact of Advanced Distributed Simulation on Readiness, Training and Prototyping*, January 1993.

[Felt95] Felton, Eric, Morrison, John, *Migratory Object Protocol and its Application to Distributed Simulation*, Proceedings of the 12th Workshop on Standards for the Interoperability of Defense Simulations, Institute of Simulation and Training, Orlando, Florida, March, 1995.

[Ferg93] Ferguson, Paula M., *Motif Reference Manual for OSF/Motif Release 1.2*, O'Reilly & Associates, Inc., Sebastopol, California, 1993.

[IEEE93] Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Std 1278-1993, *Standard for Information Technology, Protocols for Distributed Interactive Simulation*, March 1993.

[ISO84] International Organizaton for Standardization (ISO), ISO 7498-1984,, *Open Systems Interconnection - Basic Reference Model*, ISO Secretariate, Geneva, Switzerland1984.

[IST92] Institute for Simulation and Training, IST-CR-92-21, *Guidance Draft Document, Communication Architecture for Distributed Interactive Simulation (CADIS),* University of Central Florida, Orlando, Florida, November 1992.

[IST93] Institute for Simulation and Training, IST-CR-93-02, *Enumeration and Bit Encoded values for Use with Protocols for Distributed Interactive Simulation Applications*, University of Central Florida, Orlando, Florida, March 1993.

[IST93a] Institute for Simulation and Training, IST-CR-93-21, *Rationale, Communication Architecture for Distributed Interactive Simulation (CADIS),* University of Central Florida, Orlando, Florida, June 28, 1993.

[IST93 b] Institute for Simulation and Training, IST-93-40, *Standard for Distribiuted Interactive Simulation -- Application Protocols, Version 2.0 [Fourth Draft],* University of Central Florida, Orlando, Florida, February 1994.

[IST94] Institute for Simulation and Training, IST-SP-94-01, *The DIS Vision -- A Map to the Future of Distributed Simulation*, University of Central Florida, Orlando, Florida, May 1994.

[IST94a] Institute for Simulation and Training, IST-CR-94-15, *Standard for Distributed Interactive Simulation -- Communication Architecture and Security [Draft]*, University of Central Florida, Orlando, Florida, March 1994.

[IST95] Institute for Simulation and Training, IST-CR-94-15, *Standard for Distributed Interactive Simulation -- Communication Services and Profiles* [Draft], University of Central Florida, Orlando, Florida, March 1995.

[IST95a] Institute for Simulation and Training, IST-CR-95-05, *Enumeration and Bit Encoded values for use with IEEE 1278.1-1995, Standard for Distributed Interactive Simulation - Application Protocols*, University of Central Florida, Orlando, Florida, 1995.

[ITU88] International Telecommunications Union, Telecommunications Standardization Sector, *Open Systems Interconnection Model and Notation - Specification of Abstract Syntax Notation One (ASN.1), Recommendation X.208*, Melbourne, Australia, 1988.

[Levi92] Levine, John R., Mason, Tony, Brown, Doug, *LEX & YACC*, O'Reilly & Associates, Sebastopol, California, 1990,1992.

[Lope94] Loper, Margaret, L., *Phases vs. Profiles: A Strategy for CADIS*, Proceedings of the 10thWorkshop on Standards for the Interoperability of Defense Simulations, Institute of Simulation and Training, Orlando, Florida, March, 1994.

[Mace94] Macedonia, Michael R., Zyda, Michael J, Pratt, David R., Barham, Paul T., Zeswitz, Steven, *NPSNET: A Network Software Architecture for Large Scale Virtual Environments* , Presence, Vol 3. No. 4, Fall 1994.

[Mace95] Macedonia, Michael R., *A Network Software Architecture for Large Scale Virtual Environments* , Dissertation, Naval Postgraduate School, Monterey, California, September 1993.

[Milg95] Milgram, David L., *Strategies for Scaling DIS Exercises using ATM Networks*, Position Paper, 12th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, Florida March 1993.

[Milg95a] Milgram, David, Cohen, Adam, Trinko, Tom, *ExMan - Exercise Management Toolkit,* Proceedings of the 12th Workshop on Standards for the Interoperability for Defense Simulations, Institute of Simulation and Training, Orlando, Florida, March, 1995.

[Peck95] Peck, Charles C., Lander, W. Brent, Hancock John P.,*Application of Distributed Object Technology to DIS*, Proceedings of the 12th Workshop on Standards for the Interoperability of Defense Simulations, Institute of Simulation and Training, Orlando, Florida, March, 1995.

[Prat95] Pratt, David R., *Recommendation for Lower Bandwidth and Resource Requirements for DIS PDUs*, Position Paper, 12th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, Florida, March 1993.

[Ratz95] Ratzenberger, Annette C., *DIS Compliant, Interoperable, and Compatible (The Need for Definitions and Standards)*, Position Paper, 12th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, Florida March 1993.

[Rose90] Rose, Marshall T., *The Open Book - A Practical Perspective on OSI*, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1990.

[Rose91] Rose, Marshall T., *The Simple Book - An Introduction to TCP/IP-based Internets*, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1991.

[Rudi92] Rubin, Harry, *Protocol Development Success Stories: Part I*, Proceedings of the 12th International Workshop on Protocol Specification, Testing, and Verification, North-Holland Amsterdam, June 1992.

[Stal94] Stallings, William, *Data and Computer Communications*, Fourth Edition, MacMillian Publishing Company, New York, New York, 1994.

[Stev90] Stevens, W. Richard, *UNIX Network Programming*, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1990.

[Tarr94] Tarr, Ronald W., Jacobs, John W., *Distributed Interactive simulation (DIS): What is it and where is it Going?*, Presentation to the Society for Computer Simulation, 1994 Summer Computer Simulation Conference, La Jolla, California, July 1994.

[VNR93] *Encyclopedia of Computer Science*, Third Edition, Von Nostrand Reinhold, New York, New York, 1993.

[Vrab93] Vrablik, Rob, Wilbert, Deborah, , *The Use of Semi-Automated Forces to Simulate a 10,000 Entity Exercise*, Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Orlando, Florida, March 1993.

[Wayn95] Wayner, Peter, *Agents Unleashed - A Public Domain Look at Agent Technology*, AP Professional, Chestnut Hill, Massachusetts, 1995.

[Wold95] Woldt, Michael B., *DIS Correlation Tools*,Proceedings of the 12th Workshop on Standards for the Interoperability for Defense Simulations, Institute of Simulation and Training, Orlando, Florida, March, 1995.

[Zes93] Zeswitz, Steven R., *NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Interchange,* Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library — 2
   Code 013
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Chairman, Code CS — 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr Michael Zyda, Code CS/ZK — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Prof John Falby, Code CS/JA — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. Dr Don Brutzman, Code UW/BR — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

7. Major Michael Canterbury USMC — 2
   Decision Support Systems Division
   Marine Corps Tactical Systems Support Activity
   MCB Camp Pendleton, CA 92709